2009-01-01

Systematic Analysis of Unknown Integrated Circuits

Michael Brutscheck
Technological University Dublin

Systematic Analysis of Unknown Integrated Circuits

Michael Brutscheck

A Thesis Presented to the Dublin Institute of Technology
for the Degree of Doctor of Philosophy

February 2009

Supervisors:
Dr. Andreas Th. Schwarzbacher
Prof. Dr.-Ing. Steffen Becker

School of Electronic and Communications Engineering,
Dublin Institute of Technology,
Ireland.

Dedicated to my father

(1942-2005)

Abstract

In recent years the efficient and structured analysis of digital unknown CMOS integrated circuits (ICs) has attracted a lot of interest. Over the last decade different invasive and non-invasive strategies have been developed to analyse unknown ICs. However, invasive procedures must always lead to the destruction of the system under investigation. Non-invasive approaches published so far have the disadvantage that ICs are analysed using very complex and very time consuming algorithms.

The focus of this thesis is to develop a non-invasive and efficient procedure to determine fully unclassified digital CMOS ICs solely by their input-output behaviour. In this research, automata theory was used to develop algorithms to analyse and determine the behaviour of unknown ICs. A novel method to non-invasively identify the pins of the circuit using electrostatic discharge is presented. Then a division of the overall IC into different independent circuits is carried out using binary maximum sequences. Furthermore, a separation procedure was developed to quickly identify different behaviours thus, allowing to significantly minimise the following analysis. The main part of this research concentrates on the analysis of nonlinear unknown ICs which are most commonly used in ICs. Here, a novel procedure was developed capable of successfully determining the behaviour of the IC under investigation. The procedure introduced is able to automatically determine the number of states which did help to overcome of the limitations of traditional approaches. To demonstrate the correct operation of the algorithms developed a hardware analysis environment was also developed. The overall system presented in this research was implemented and tested using the IEEE ISCAS benchmarks. For every circuit analysed the behaviour was successfully determined. However, the methods developed in this research are not limited to the investigation of unknown CMOS ICs. They can also be used to non-destructively analyse structures, functions or behaviours in a wide variety of areas such as lexical analysis, pattern matching, communication protocols or software analysis.

I certify that this thesis which I now submit for examination for the award of Doctor of Philosophy, is entirely my own work and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my work.

This thesis was prepared according to the regulations for postgraduate study by research of the Dublin Institute of Technology and has not been submitted in whole or in part for an award in any other Institute or University.

The work reported on in this thesis conforms to the principles and requirements of the Institute's guidelines for ethics in research.

The Institute has permission to keep, to lend or to copy this thesis in whole or in part, on condition that any such use of the material of the thesis be duly acknowledged.

Signature Michael Brutscheck Date 01/04/2009

Michael Brutscheck

Acknowledgements

Firstly, I wish to express my sincere gratitude to my supervisors, Dr. Andreas Schwarzbacher and Prof. Steffen Becker, for their support and help throughout this thesis. Their advice and guidance was invaluable on many occasions.

I would also like to thank Prof. Bundschuh for his support and help both for technical and spiritual advice throughout the years.

Furthermore, I want to acknowledge the support of my friends and colleagues both inside and outside the department, especially Marco, Benjamin, Erhard, Steffen and Martin.

Additionally, I would like to acknowledge the funding received through rector's office and the Department of Computer Science and Communication Systems of University of Applied Sciences Merseburg, Germany, especially Prof. Zwanziger and Prof. Hartmann.

I also wish to thank my parents for the support they have given me over all the years. Without their help it would be impossible for me to stand here.

Finally, and most of all, I would like to thank my wife Silke for her understanding and support, and for keeping me grounded over all the years. Thanks to our sunshine Jamie for cheering me up. Everything is going to be alright!

Abbreviations List

ACF	Autocorrelation Function
ALM	Adaptive Logic Unit
ASIC	Application Specific Integrated Circuit
ATPG	Automatic Test Pattern Generation
BiCMOS	Bipolar Complementary Metal Oxide Semiconductor
BISR	Built-In Self-Repair
BIST	Built-In Self-Test
CCF	Cross-Correlation Function
CCS	Current Compared State
CIFS	Current Investigated Following State
CIP	Current Investigated Path
CIS	Current Investigated State
CMOS	Complementary Metal Oxide Semiconductor
CNo	Combination Number
CPU	Central Processing Unit
DFT	Design for Testability
EDIF	Electronic Design Interchange Format
ESD	Electrostatic Discharge
FA	Finite Automaton

FNoIW	Found Number of Input Words
FNoOW	Found Number of Output Words
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
GaAs	Gallium Arsenide
GF	Galois Field
GND	Ground
HMB	Human-Body-Model
IC	Integrated Circuit
ISCAS	International Symposium on Circuits and Systems
IW	Input Word
JTAG	Joint Test Action Group
LFSR	Linear Feedback Shift Register
LSB	Least Significant Bit
MIMO	Multi-Input Multi-Output
MM	Machine-Model
MOS	Metal Oxide Semiconductor
MOSFET	Metal Oxide Semiconductor Field-Effect Transistor
MSB	Most Significant Bit
MSI	Medium-Scale Integration
MSIW	Most Significant Input Word

NoAIS	Number of Apparently Identical States
NoCycl	Number of Cycles
NoDS	Number of Distinguishable States
NoE	Number of Entries
NoFF	Number of Flip-Flops
NoFS	Number of Found States
NoGS	Number of Guessed States
NoIB	Number of Independent Blocks
NoIP	Number of Input Pins
NoIW	Number of Input Words
NonDS	Number of Not Distinguishable States
NoOP	Number of Output Pins
NoUIP	Number of Used Input Pins
NoUOP	Number of Used Output Pins
NoS	Number of States
NoSP	Number of State Paths
NoTr	Number of Trees
OF	Output Function
OW	Output Word
OWC	Output Word Combination
PC	Personal Computer

pn	Pseudo Noise
RAM	Random Access Memory
RIE	Reactive Ion Etching
SSI	Small-Scale Integration
STT	State Transition Table
UI	User Interface
VCC	Supply Voltage
VHDL	Very High Speed Integrated Circuit Hardware Description Language
VLSI	Very Large Scale Integration

Table of Contents

List of Figures

List of Tables

1 Motivation

The semiconductor revolution was started by the inventors William Shockley, Walter Brattain and John Bardeen on 23rd December 1947 with the first successfully tested point-contact **trans**fer re**sistor** (transistor) [Bard50] and [Shoc51]. In the following years the investigation of suitable semiconductor materials used to integrate electronic circuits became a major research objective. In 1958 Jack Kilby created the first integrated circuit at Texas Instruments proving that resistors, transistors and capacitors could exist on the same semiconductor material [Kilb76]. Only one year later Robert Noyce from Fairchild provided the final piece of the puzzle. He produced the first chip based on planar Silicon technology [Klin04]. In 1965 Gordon Earl Moore expressed Moore's Law, which anticipates the doubling of circuit density and capacity of semiconductor devices every eighteen months or a quadrupling every three years [Moor65]. The accuracy of this prediction has been shown over the last four decades [Moor06]. Today, Moore's following citation is more than ever an issue.

"Integrated circuits will lead to such wonders as home computers, automatic controls for automobiles, and personal portable communications equipment." – Gordon Earl Moore, then Director of Research & Development, Fairchild Semiconductor Division, 1965 [Moor65].

The essential driving forces of the semiconductor industry are increasing complexity, cost reductions of about 25% per annum, power consumptions, speed and integration of additional functions. Today, billions of transistors together with other semiconductor devices can be integrated on a single chip [Klin04]. The present technology enables the production of increasingly larger circuits on increasingly smaller space. Through this integration it is possible to provide extensive functionality on a single chip. Nowadays, nearly all consumer and industrial applications are based on integrated circuits helping to increase the standard of living. Additionally, in many cases integrated circuits (ICs) facilitate the technical realisation of systems which would be otherwise too expensive, too complex or

which would consume too much power. Integrated circuits are used in both analogue and digital system design. In analogue integrated circuits many different amplifiers, filter circuits, sensors and actuators were implemented [Raza08] [Sedr04]. In contrast digital ICs contain digital state processes and logical linking resulting from digital technology.

Beside Bipolar, BiCMOS and GaAs, CMOS technology exists which is the most popular and dominant technology for the implementation of digital systems since the early 1970s [ibid]. This is because of its small size and ease of fabrication. Therefore, for the analysis of unknown ICs CMOS technology was chosen in this research. Following Moore's Law the characteristic size which is the minimum channel length of the MOS transistor has decreased rapidly over the last years. This has resulted in a test chip with a published channel length of 32nm on a functioning 300mm wafer [Knap08]. The low power dissipation of MOSFETs inside the IC enables extremely high levels of integration of both logic and memory circuits. Thus, very tight circuit packaging and very high levels of integration are possible. Today, different levels of integration are available. On one hand, small-scale integrated (SSI) circuit packages contain one to ten logic gates and medium-scale integrated (MSI) circuit packages contain 10 to 100 gates per chip. These integration levels are useful for assembling digital systems on printed-circuit boards. On the other hand the most important application of CMOS technology is very large scale integrated (VLSI) logic with millions of gates per chip. In some applications different CMOS integration scales are used. Furthermore, the high input impedance of the MOS transistor is another advantage of CMOS technology. Thus, the temporary storage of information in both logic and memory circuits is possible.

The complex structure and thus, the production of CMOS ICs resulted in the demand for easily and efficient operated test and analysis tools. These tools ensure high quality output during the manufacturing of integrated circuits and its following inspection towards their correct operation. However, other applications also require the detailed knowledge about the function of an IC. In these cases, it is for example necessary to monitor patent rights of different manufacturers or to obtain additional information of a system not described in data books. Furthermore, an analysis of unknown ICs might be required when the label is lost or it is necessary to find out

more about the internal structure of the integrated circuit. It is also conceivable that an integrated circuit will become obsolete. Here, the IC's structure then might be read out and can be implemented with potentially added functionality into a new IC to reuse old functions. In this context ASIC to FPGA and FPGA to ASIC conversions have also become increasingly important over the past decade [Tekm09] [Atme09]. Normally, the net list of the ASIC or FPGA to be converted is known and a conversion into the FPGA or ASIC can be accomplished using traditional methods. In case of lost information or unavailability of information other solutions must be found. Here, the non-destructive analysis of the input-output behaviour of the IC investigated might help to solve such problems. Further applications are conceivable in a wide range of IC design for example in simulation tools where the function of an unknown simulated model has to be determined. Hence, it is not surprising that various test and analysis procedures have been developed over the last years. However, both analysis as well as test procedures use rather similar technology. The important difference is that test tools already know the correct transition or internal function before investigating the circuit. Furthermore, the important information from a test tool is whether the IC works properly or not, which means it simply passes or fails the test. At first a general overview of current test procedures for ICs will be given. Traditional invasive and non-invasive procedure will be given in Section 1.2. Section 1.3 then will provide an overview of the overall thesis.

1.1 Test Procedures

There are several possibilities to support test tools in advance. Firstly, the IC designer may implement specific hardware to test the IC. These are for example design-for-testability (DFT) with fault detection [Lu05], built-in self-test (BIST) [Naka06], scan test [Wang06] and built-in self-repair (BISR) [Alsa06]. Starting with built-in self-test (BIST) an integrated circuit contains an internal test circuit which generates test sequences [Char03]. Furthermore, these test sequences are compared to reference values which results are transmitted to the test environment. Over the years BISTs have become easier to implement. However, these test circuits need little space and thus, it is a cost and time effective procedure. BISTs are also used to prove the correct function of a processor during its power-on and power-off procedure to detect malfunctions and to avoid complications. There are different

types of built-in self-tests, for example processor, memory, analogue and mixed signal BIST as well as boundary scan test. The boundary scan test is one of the most important test procedures and will be described in the following. In 1990 the Joint Test Action Group (JTAG) defined the test standard IEEE 1149.1 [JTAG09] also known as boundary scan test which can be used to test ICs after manufacture. Figure 1.1 shows the typical structure of a boundary scan IC [Auer96].

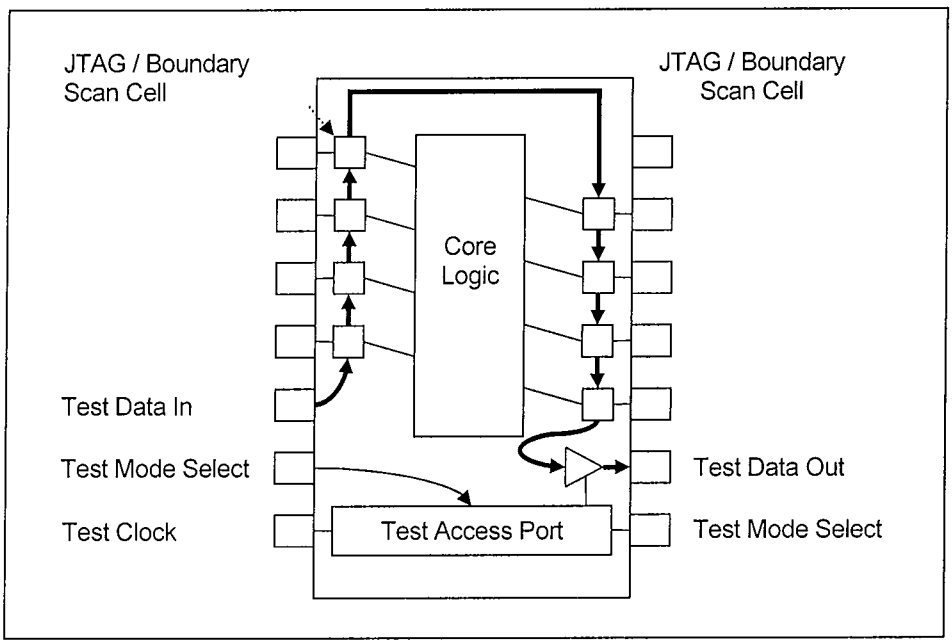

Figure 1.1: Simplified Boundary Scan Structure

As can be seen in Figure 1.1 additional to the core logic some additional test logic is implemented in the integrated circuit. Here, it is possible to store significant circuit states in a shift register, also named boundary scan cells, and to read it out serially after the test. Using boundary scans on the chip level a large portion of possible malfunctions can be detected [ibid]. The same procedure can be applied at the board level. Here, it is possible to detect most of the stuck-at faults, short-circuits or open wires [ibid]. Another cost and time effective test procedure is the built-in self-repair (BISR) procedure [Char03]. This procedure is often implemented in combination with memories. Here, large memories and high-end memory technologies may lead to an increasing number of defects. Using BISR solutions the defective part of the memory to be tested can be replaced by redundant blocks which are free of defects. If the memory contains critical contents it is possible to disable the defective blocks

and replace them without losing the original memory content. After having described traditional test procedures analysis procedures will discussed in the next section.

1.2 Analysis Procedures

Analysis is the process to obtain information about the internal structure and function of the IC investigated. In contrast to the testing of integrated circuits analysis procedures do not require any prior knowledge about the internal structure of the IC. However, analysis tools support the knowledge intensive process of finding an answer of hidden functions and structures in the ICs investigated [Jarz95]. In this context the analysis will also be described as reverse engineering [Chik90]. Until now, several methods for reverse engineering have been developed and will be presented next. Traditionally, invasive procedures were used to obtain the internal function of the IC analysed. However, confidential data of ICs exist which have to be protected against competitors. Therefore, companies are reluctant to publish reverse engineering procedures. Nevertheless, in the following a general overview of the most important both invasive and non-invasive procedures published will be discussed.

1.2.1 Invasive Analysis Procedures

Basically, the traditional invasive analysis of unknown ICs consists of four main steps which are given in Figure 1.2 [Ixen09].

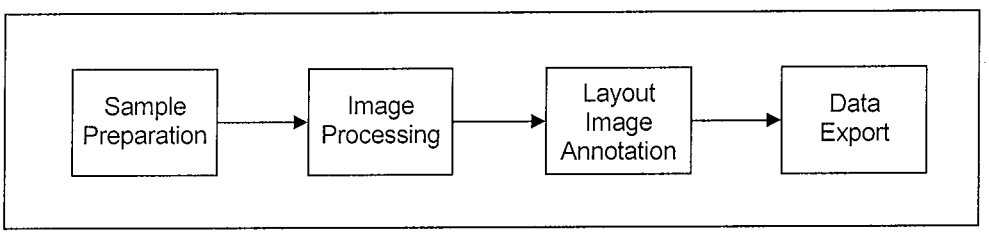

Figure 1.2: Reverse Engineering of ICs in General [ibid]

The sample preparation is the first step followed by the raw die image processing which recognises cells as well as interconnections. After an electrical design rule check the layout image is recorded. From this the schematic is automatically generated in the last step. This includes the data export in various formats such as Verilog, VHDL or EDIF. As can be seen in Figure 1.2 the sample preparation is the

first step of the overall analysis procedure. Here, highly precise machines peel the integrated circuit layer by layer to obtain information about the silicon structure of the IC under investigation. A delayered IC can be seen on the right photo in Figure 1.3 where the raw die, bonding and pads are dissected.

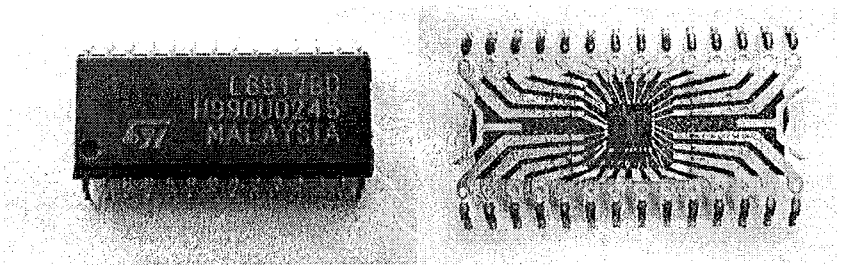

Figure 1.3: Original (left) and Delayered IC [ibid]

Delayering is the process in which various layers are removed from an IC to be investigated [Nise09]. This process was developed many years ago. However, it has been an imperfect procedure. Reactive ion etching (RIE), wet chemistry and mechanical polishing are the three most important methods currently used [ibid]. Each of these methods has its own drawbacks. The first method reactive ion etching is only able to etch certain materials. Furthermore, while these etches tend to be selective the overall RIE process works anisotropic which can be a disadvantage. The second method used is wet chemical etching. It is a selective procedure but difficult to control. Therefore, it will often result in etching lower metal layers of the IC under investigation. The last method often used is mechanical polishing. In contrast to wet chemical etching mechanical polishing is not very selective. Moreover, it rounds the edges of the sample being delayered [ibid]. Furthermore, each material which appears on top is polished. This is a big disadvantage.

After the sample preparation the next step uses photographic and image-processing systems as well as software tools to determine the function of the unknown integrated circuit [Blyt93]. This is also called optical inspection of the raw die. An example of a raw die image can be seen in Figure 1.4.

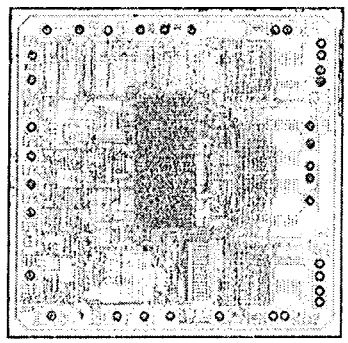

Figure 1.4: Raw Die Image of a Metal Layer [Ixen09]

Several optical inspection systems were introduced in the past. However, Bourbakis presented a knowledge-based expert system for VLSI reverse engineering procedure to inspect the IC on the wafer optically [Bour02]. Figure 1.5 describes the image processing in principal.

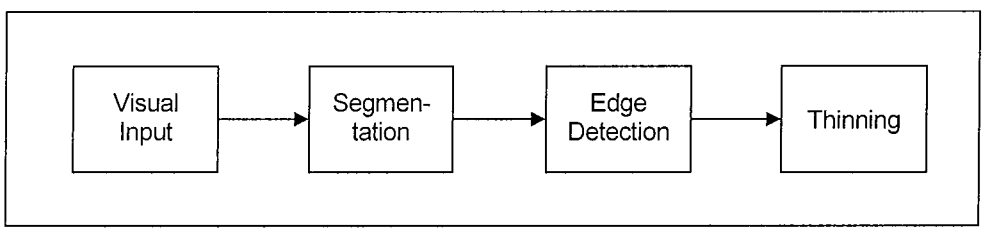

Figure 1.5: Image Processing Sequence for Shape Extraction [ibid]

The system inputs are usually raw die images as can be seen in Figure 1.4. The output is the functional behaviour of the entire system. First, the system starts by scanning the VLSI image. Next, the segmentation separates the component colours from the layout drawing. Edge detection then extracts the blocks for each layer of the layout. Finally, thinning simplifies the abstraction of the transistor level representation. Here, the advantage of the system described is that no test needles are necessary. Normally, tungsten needles are used which have a size of about 12 to 35µm. To make a good contact between the test adapter and the wafer on the silicon aluminium or copper pads have to be placed on the IC which have a size of about 60µm x 60µm. A lot of space is required to place many of such pads which cannot be used for active parts of the IC. However, if the layout is given the analysis is easily accessible. This is possible because no package is mounted around the silicon. However, due to the rapid reduction of the minimum channel length of the MOS transistor as well as the increasing layer numbers in an IC the traditional reverse engineering has several drawbacks. First, the etching process will become less

accurate with decreasing feature sizes. This results in an increasing number of sample damages. Furthermore, optical inspections are limited by the resolution of the microscope used which again is becoming more limited by smaller technologies. Moreover, companies are developing casting compounds with high stability against etching to protect the IC against reverse engineering [Schä08]. Therefore, it will become more and more difficult to peel the IC housing without damage to the metal layers. Furthermore, several layers are currently implemented in an integrated circuit. In the future three-dimensional circuit elements will be used [Patt06] which will lead to a further hindrance of traditional reverse engineering. In summary, the obvious disadvantage of traditional analysis procedures engineering procedures described is the destruction of the integrated circuit to be analysed. Therefore, the IC cannot be reused. After having described the traditional invasive analysis procedures the traditional non-invasive analysis methods are explained in the next section.

1.2.2 Non-Invasive Analysis Procedures

The non-destructive investigation of integrated circuits has become one of the most important fields of reverse engineering research [Hans99], [Bour02] and [Hans09]. Although the identification of unknown integrated circuits is a difficult problem it has been studied in different disciplines and at various times [Lee96]. In all cases a black box is given whose structure has to be identified from its input-output behaviour only. Moore first proposed the identification problem and provided an exponential algorithm [Moor56]. He showed that the problem of machine identification is inherently exponential. In contrast to FSM identification the purpose of conformance testing is to find out if an IC implementation is different than its specification. This kind of testing is also called fault diagnosis [Frie71]. In general, this is an identification problem. However, if it is possible to constrain the number or type of faults that can occur, then several efficient methods exist to solve part of the problem [ibid], [Gone80] and [Lu05]. Generally, for the identification a FSM A is given and a test sequence has to be found, which can determine the transition diagram of A from its input-output behaviour. Such procedures can also be applied to reverse engineering of communication protocols [Lee93] where the behaviour of the implemented protocol standard is observed. First, a certain a priori knowledge of the finite state machine A is necessary, because without any assumptions it is impossible

to solve the problem. Moore therefore, defined that A has to be strongly connected, reduced and does not change during the analysis. Moreover, the input and output alphabet as well as the upper bound r on its state numbers are known. The reasons are akin to conformance testing of sequential circuits. The assumption that A is reduced is important to have the identification problem uniquely defined, because an experimental approach cannot differ between equivalent machines. The investigation of FSMs attracted little interest until a few years ago [Lee96]. Fault detection is now being studied anew due to its applications when testing communication protocols [Holz90]. Here, a number of methods have been published on conformance testing such as [Gone80], [Aho91], [Kats91], [Mill91] and [Bar92].

Neural networks are another possibility to investigate unknown ICs [Lack97]. The main advantage of neural networks is the potential to classify known structures in a fast and efficient manner. However, this method has also several disadvantages. The network has to be divided and structures have to be taught. A nearly endless amount of preliminary classifications would be necessary to describe an IC of variable size using a neural network. If only one part of the structure is to be determined by the network it would be possible to make some assumptions but it would be impossible to define it exactly. Therefore, a more complex further investigation of the system would be necessary. Due to these disadvantages a neural network is not an efficient solution for the problem. In 2004 Kuroe presented a new architecture of neural networks called recurrent hybrid neural networks for representing deterministic finite state automata [Kuro04]. This recurrent hybrid neural network consists of two types of neurons, static neurons and dynamic neurons. Moreover, the proposed model of the neural network is capable of precisely representing FSMs with a network size smaller than the existing models. Furthermore, Kuroe proposed an identification method for FSMs from a given set of input and output data by training these neural networks. Here, a data set of input and output sequences of a target FSM were given. Then, the input and output relation have to become equal to the FSM. In this context the parameters of the neural network have to be determined. Therefore, the following assumptions are made. The set of state symbols, the initial state, the state transition function g and the output function f of a finite state machine are unknown. However, the set of input symbols $\underline{X}$

and output symbols $\underline{Y}$ of FSM are known. A set of data of input sequences $\{x(t)\}$ and the corresponding output sequences $\{y(t)\}$ are available. It should be noted that the proposed neural network is easy to be identified because of lesser number of learning parameters, which results from the smaller network size. In their paper the authors also assume that the number of state symbols is estimated to be less than five ($r < 5$) [ibid]. This is a highly simplified assumption and cannot be used in a realistic environment.

All non-invasive procedures proposed so far have several disadvantages for realistic IC analysis. Moreover, all algorithms described lack the knowledge of the input-output alphabet. Therefore, it is important to know where the inputs, outputs, supply voltage and ground pins are located. This is indispensable for an analysis of any real unknown IC. However, the overall research objective of this thesis is to show how the internal functions and structures of unknown CMOS integrated circuits can be efficiently determined in a non-destructive manner. In particular, this thesis describes a novel method to non-invasively analyse digital unknown CMOS integrated circuits. After presenting traditional non-invasive procedures the thesis overview will be presented in the next section.

1.3 Thesis Overview

The remainder of this thesis describes a novel method for the non-invasive and systematic analysis of unknown digital CMOS ICs. Firstly, the theoretical background and the general models of automata theory will be provided in Chapter 2 for a better understanding of the overall thesis. Furthermore, all related parameters will be described which are required to characterise unknown ICs mathematically. This is important to fully understand the analysis procedures presented in the following chapters. Furthermore, the automata synthesis and analysis will also be introduced in Chapter 2. Here, the relationship of both the synthesis and the analysis is the basis to abstract unknown integrated circuits. Moreover, the traditional classification of automata will be explained which is important to allow a simplification of the circuit analysis. Chapter 3 then focuses on the analysis process to investigate different finite state machines. First, a general overview of the overall analysis procedure developed will be given which consists of three main parts. First,

the determination of pin types and their location will be explained which automatically identifies the most common pin types of a digital CMOS ICs. Secondly, the separation procedure of unknown ICs will be described which classifies the automata in the beginning of the analysis procedure. This has the advantage that depending on the behaviour found the following analysis procedures can be significantly reduced. Finally, the identification procedures used to report combinatorial or linear sequential behaviour will also be explained in Chapter 3. Chapter 4 then describes the identification process in detail for nonlinear behaviour which is the third main part of the overall analysis procedure. However, a large number of unknown ICs exhibit nonlinear behaviour. Therefore, the main focus of Chapter 4 is to find new strategies to non-destructively identify nonlinear deterministic finite automata. To identify unknown ICs traditional procedures always require prior knowledge of the number of internal states. This proposed novel procedure is able to identify the IC under investigation without any such prior knowledge. After having mathematically explained the different parts of the overall analysis procedure in Chapter 3 and Chapter 4 the results of this research are then be discussed in Chapter 5. To demonstrate the correct operation of the procedures developed an analysis environment was designed and is introduced in Chapter 5. Using this analysis environment benchmark models as well as user defined models of real ICs are investigated. Using these real ICs the results of the identification procedures will be evaluated in Chapter 5. In Chapter 6 the conclusions of the research are presented and possibilities for further enhancements of the analysis procedure are proposed.

2 Automata Theory

The previous chapter gave an overview of the motivation for the investigation of novel non-invasive analysis procedures. Furthermore, traditional invasive as well as non-invasive test and analysis tools were discussed. This chapter now describes the fundamental background and the parameters of automata to understand the analysis procedures investigated in the following chapters. First, automata synthesis and analysis will be introduced which is the basis to describe unknown ICs. Section 2.2 then shows the general model of finite state machines including the input parameters, the internal states, the output states and the relationship between them. Afterwards, the traditional classification of automata is explained which results in deterministic finite state machines as will be described in Section 2.4. Further, the classification is important to simplify the further analysis and is applied to determine the unknown behaviour of the IC under investigation. Section 2.5 describes the most common models which are used in automata theory. In the last section the description forms of finite state machines are given which are relevant to mathematically characterise the unknown IC.

2.1 Overview of Synthesis and Analysis

Automata theory describes the behavioural model of systems and is an integral component of present theoretical computer sciences, mathematics and signal and system theory [Koha78] [Wuns93]. Furthermore, it is possible to transfer any given problem into an automaton structure [Wuns86]. However, the definitions of automata models are universal and are used to solve different problems. Therefore, analogue as well as digital integrated circuits can be described using automata models. Digital structures become increasingly significant and are often used in the development of automata structures, because these ICs can be easier and faster developed than their analogue counterparts. These integrated structures are described with the help of computational programs, this process is called synthesis. In contrast to synthesis, the analysis of integrated structures describes the reverse procedure. Analysis determines the mathematical functions as well as the technical models of

the hardware automata. Analysis works with several descriptions of the results obtained. Hence, the analysis of an unknown automaton is a particularly difficult procedure. This is because the number of internal state parameters and mathematical functions of the integrated circuit are usually unknown. In automata theory synthesis as well as analysis are the two most important methods and the parameters used for the description of digital systems can be seen in Figure 2.1.

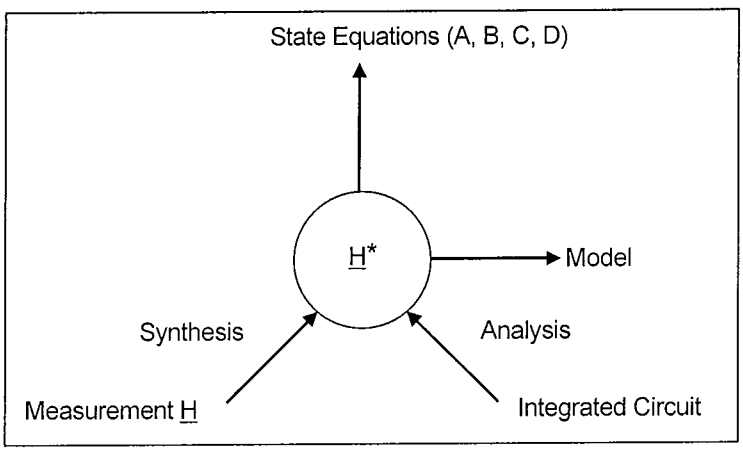

Figure 2.1: Realisation Scheme of Analysis and Synthesis

In Figure 2.1 a given system characteristic is defined mathematically by $\underline{H}^*$ [Wuns93]. The system characteristics and the operation of a system are illustrated by $\underline{H}^*$. The model and state equations are suitable representation forms of the behaviour model. In the following sections both the synthesis and the analysis are described.

2.1.1 Synthesis

In general, it can be said that synthesis constructs are a model for a given system characteristic $\underline{H}^*$ [ibid]. Here, synthesis describes the conversion of a given or required behaviour into hardware. Furthermore, from the mathematical system the realisation of digital hardware on the CMOS integrated circuit will be received. For the definition of the structure the mathematical function is not solely sufficient in this case. Here, statements about the available components, about set structural intentions and optimisation criteria must be available [Boch81]. However, first the user characterises the future functions and tasks of the mathematical model with description languages, for example VHDL or Verilog [Dewe97]. Furthermore, a compiler transfers the described functions into hardware components. The result of

the synthesis leads to the implementation of the mathematical behavioural model into a hardware structure [Reic01] where the structure can be different when using different compilers. The comparison of the theoretical model with the real development hardware structure is the last step of this procedure. However, the function of the synthesis was further refined and perfected over the last years [Dewe97], [Klin04] and [Raza08]. After having described the synthesis in the next section the analysis will be discussed.

2.1.2 Analysis

The system analysis of CMOS integrated circuits is concerned with the systematic investigation of an already existing structure. System theory describes the internal structures with the help of analysis. The analysis can be used in many areas. In this thesis it is applied to the determination of unknown integrated circuits. Here, the analytic process considers the unknown IC as a black box. The properties and operation of the black box are examined in the course of the analysis procedure to create a model which corresponds to the black box. Furthermore, the object examined is dissected and after that disassembled into its components. Then, the components will be arranged, examined and evaluated. The model created is always a restricted, reduced, abstracted copy of reality. However, for this analysis usually only the input and output pins as well as the package type are available. The result of such an analysis can be either formal methods or graphical methods [Aise67]. In addition, the analytic process will be considered mathematically on the basis of a theoretical model. Moreover, on the basis of these findings the analysis will be processed using real systems. In this research CMOS integrated circuits will be considered as automata and especially as finite state machines. Since the automata theory is a diversified complex field, many different categories of automata types are investigated. Afterwards, certain types of automata are separated. Furthermore, these findings of automata can be used to characterise CMOS integrated circuits. In detail, the automata theory deals with finite state machines and the CMOS integrated circuits are generally composed of such finite state machines. The coherences between the mathematical automata model and the integrated circuits can be used to find a new approach to determine unknown systems. In Figure 2.2 it can be seen that

the functions and structures of the automata models can be transferred to the integrated circuits.

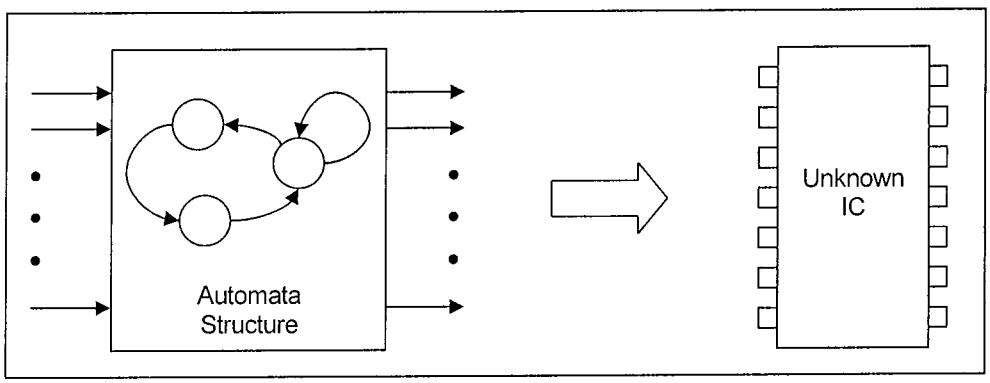

Figure 2.2: Abstraction of an IC to a Finite State Machine

This means, that the same tasks can be executed using the finite automata model as well as an integrated circuit. Therefore, the internal structure processes the input values in same manner as the original IC. Because of that, it is possible to describe the internal structure of the IC using automata structures [Balk93] which is used to investigate unknown ICs. Moreover, in this thesis different automaton types are investigated and are analysed with respect to the operating method of finite state machines. Here, the investigation of finite state machines will help to determine unknown ICs. The next section will discuss the modelling of finite state machines.

2.2 Modelling of Finite State Machines

Automata are abstract models and used in theoretical computer sciences to analyse and prove definite properties of problems and algorithms [Lunz06]. In this thesis, the automata are reduced to their fundamental properties. This simplification of the attributes allows an understanding of the behaviour and the structures of automata. In computer sciences automata are confined to digital systems [Hopc02]. Models in digital technology can be characterised in the field of hardware and software with the help of digital automata systems. In this case, the automata have fundamental practical importance for discrete systems. The automata models are described with the input vector $\underline{X}$ called input word from the input alphabet Σ, with the internal state vector $\underline{Z}$ and the output vector $\underline{Y}$ called the output word as illustrated in Figure 2.3.

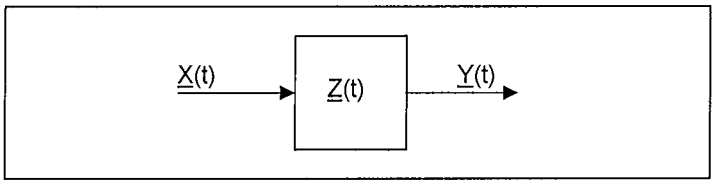

Figure 2.3: Behavioural Model of an Automaton

The parameters of digital automata $\underline{X}(t)$, $\underline{Z}(t)$ and $\underline{Y}(t)$ are discrete vectors and carriers of time variable information. The general behavioural model of automata can be explained in simple terms. Different parameters are applied at the input $\underline{X}$. The input parameters can have many different values $x_1, x_2, \ldots, x_k$ which are abstracted to $\underline{X}$. This input $\underline{X}$ can cause a change of the internal states $\underline{Z}$ and therefore, a change of the output values $\underline{Y}$. The number of internal states $\underline{Z}$ may also consist of several values $z_1, z_2, \ldots, z_l$. The internal structures of the states describe the mode of operation of different automata. The complete mode of operation of an automaton model depends on the given input parameters, the number of internal states, the structure of the stages as well as on the output parameters. The output $\underline{Y}$ is an initial vector, whereas the number of vector components corresponds to the number of outputs $y_1, y_2, \ldots, y_m$. Automata are subdivided into different classes to be able to describe various functions and operations. Fundamentally, deterministic and nondeterministic automata are distinguished in automata theory [Lunz06] as will be described in the following section.

2.3 Fundamental Classification of Automata

Automata are applied in various areas of science and technology to describe different functions and operations. Therefore, cognitions about the internal structures are required and are used to solve diverse problems with the help of automata theory. The applications of automata are diverse. However, for the analysis only certain automata types must be examined. Here, the determination of internal structures of different FSMs is used for particular areas of automata types. In principle, an automaton is a mathematical system, which can be used in many areas to describe a given task [Hopc02]. The especially important distinguishing feature in automata theory is whether it is a deterministic or a nondeterministic automaton. This property essentially determines the further divisions and applications of automata types. It can

be seen in Figure 2.4 that the deterministic and nondeterministic automata are a sub class of general automata.

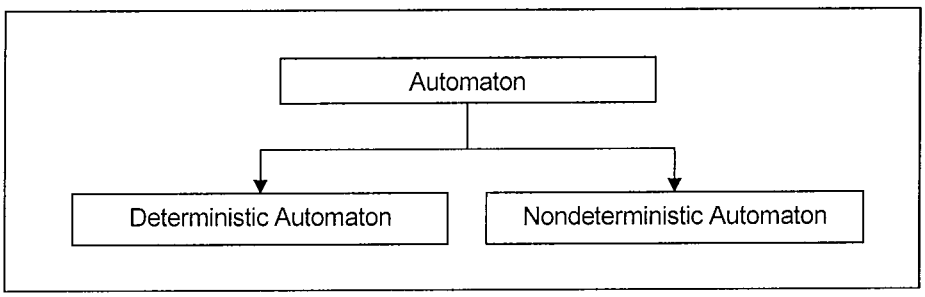

Figure 2.4: Division of Automata

Deterministic digital systems describe automata with the following properties. A deterministic system has a finite number of input and output signals with a finite number of input and output values. This is also called finite input and output alphabet. Furthermore, this deterministic system works within time intervals and the description of the system behaviour can occur in a finite amount of time. In addition, the output values can be defined at every time by present and past input values. Thus, every change of the input values $\underline{X}(t)$ leads to a defined next internal state $z(t+1)$. If the deterministic automat is located at some state, the input value and the present internal state determine the exact transition to the next internal state. If the input values $\underline{X}(t)$ change, only one following states is possible. This means that an input combination of the input alphabet Σ leads to a specific change of the internal states. Furthermore, the same combination of input values $\underline{X}(t)$ and the same current internal states always lead to the same next internal state as can be seen in Figure 2.5.

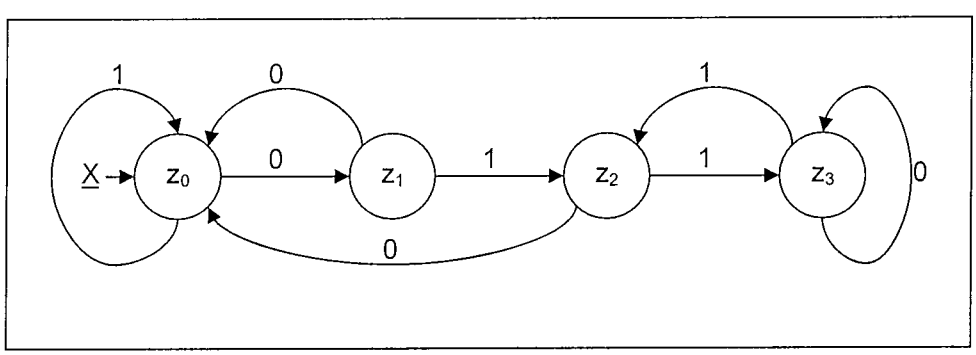

Figure 2.5: Transition Diagram of a Deterministic Automaton

It can be seen in Figure 2.5 that every state can be stimulated using two different input values. For example, the state z_1 changes from z_1 to z_2 with the input value '1'

and from state z_1 to z_0 with the input value '0'. It is possible that the current state changes to another state. Otherwise, the state does not change. The states can only respond to two different values. This is possible, because the automata considered are binary systems. The input and output alphabet only consist of '0' and '1'. In any given situation, nondeterministic automata have the possibility to select from many different following states as presented in Figure 2.6.

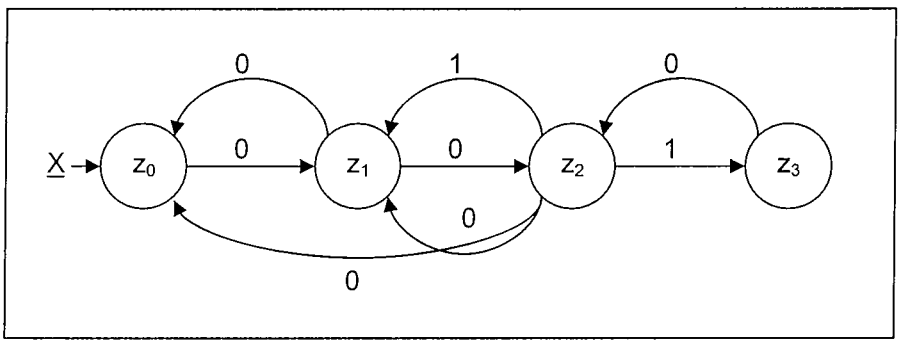

Figure 2.6: Transition Diagram of a Nondeterministic Automaton

With every step the automaton reads an input combination. However, for a defined current internal state and a given input combination none, one or more than one legal succession state can exist. In accordance with an input several output values are permissible. For example during state z_2 the input value '0' might cause two different following states z_1 or z_0. Therefore, if the input value '0' occurs it is not possible to determine the following state z_2. The selection is random and cannot be predicted a priori. The mode of operation of a nondeterministic automaton is particularly difficult because the acceptance of the input values is algorithmically irreproducible through the present automaton. The advantage of the nondeterministic finite state machines is a simpler synthesis with set mathematical models [ibid]. However, deterministic and nondeterministic automata can be further separated which are for example finite state machines [Laws04], Turing machines [Hopc02], pushdown automata [ibid] and register machines [Shep63]. However, integrated circuits considered in this thesis are deterministic finite automata and will be described in the next section.

2.4 Finite State Machines

The number of possible input parameters $\underline{X}(t)$, internal state parameters $\underline{Z}(t)$ and output parameters $\underline{Y}(t)$ are infinite for fundamental digital automata. Therefore, automata analysis is simplified because the parameters are subjected to the following conditions: Automata analysis is concerned with binary automata. In this case, the input alphabet consists of the amount of {0, 1} values. This amount is described as a state space or a Galois field (GF). A Galois field is a finite field with finite number of elements [Wuns93]. Therefore, binary automata are often described by $GF(2)$ with the finite amount {0, 1} and are generally denoted as binary finite state machines. The input alphabet, the number of possible value assignments of $\underline{X}(t)$, $\underline{Z}(t)$ and $\underline{Y}(t)$ are finite. In this case the automata are denoted as finite automata (FA) or as finite state machines (FSM). A finite state machine is a subgroup of the deterministic or nondeterministic automata as illustrated in Figure 2.4. Furthermore, it can be said that an automat is called finite automaton, if it consists of a finite amount of internal states $\underline{Z}$ and transitions. The finite automata can be fully defined by the internal state of the system with every clock signal. Therefore, a finite automaton has a finite number of internal states and state transitions. The finite state machines are special cases of deterministic automata. They are used in general applications and especially in the development of digital circuits, modelling of the application behaviour in software technology as well as word and language identification. They are applied as behavioural models and consist of states, state transitions and actions. A state stores the information of the past. This means, that it specifies the changes of the input sequence starting at the system start. A state transition is a change of the state of the finite state machine and is described by logical conditions which must be fulfilled. The output of the finite state machine is an action which occurs in a certain situation. Furthermore, by the respective input behaviour and the internal states the finite state machine proceeds with discrete times t_x where ($x = 1, 2, 3, \ldots$) into a new internal state and an unequivocally definable output word. The deterministic and nondeterministic finite state machines can be assigned generally by the specification of the quintuple as presented in (2.1).

$$A = \left(\underline{X}, \underline{Z}, \underline{Y}, g, f\right) \tag{2.1}$$

The quintuple is defined by the finite amount of input values $\underline{X}$, of internal states $\underline{Z}$, the initial values $\underline{Y}$, the result function f and the transition function g. The transition function is given by $g(x, z)$ and it is represented by (2.2).

$$g : \underline{Z} \times \underline{X} \rightarrow \underline{Z} \tag{2.2}$$

The result function can be described by $f(x, z)$ or $f(z)$ and is illustrated by (2.3).

$$f : \underline{Z} \times \underline{X} \rightarrow \underline{Y} \; or \; f : \underline{Z} \rightarrow \underline{Y} \tag{2.3}$$

The function of the finite state machines is given by the automata which read word by word of the input alphabet Σ. Thus, they can change into the next state. The transition function as presented in Equation (2.4) dictates which state has to be accepted.

$$g : \underline{Z} \times \underline{X} \rightarrow \underline{Z} \tag{2.4}$$

At the end of the calculation it is decided whether the read word is accepted or rejected. For the description of finite state machines only deterministic finite state machines are considered. This is possible because finite state machines in unknown CMOS integrated circuits respond to input words or input vectors from the alphabet Σ and the present internal state is transferred into another determined state. For every input word $\underline{X}$ and current internal state exactly one state transition exists. Therefore, the deterministic finite automata must be considered only for the determination of finite state machines in digital systems. Figure 2.7 shows the overall classification of automata of deterministic automata.

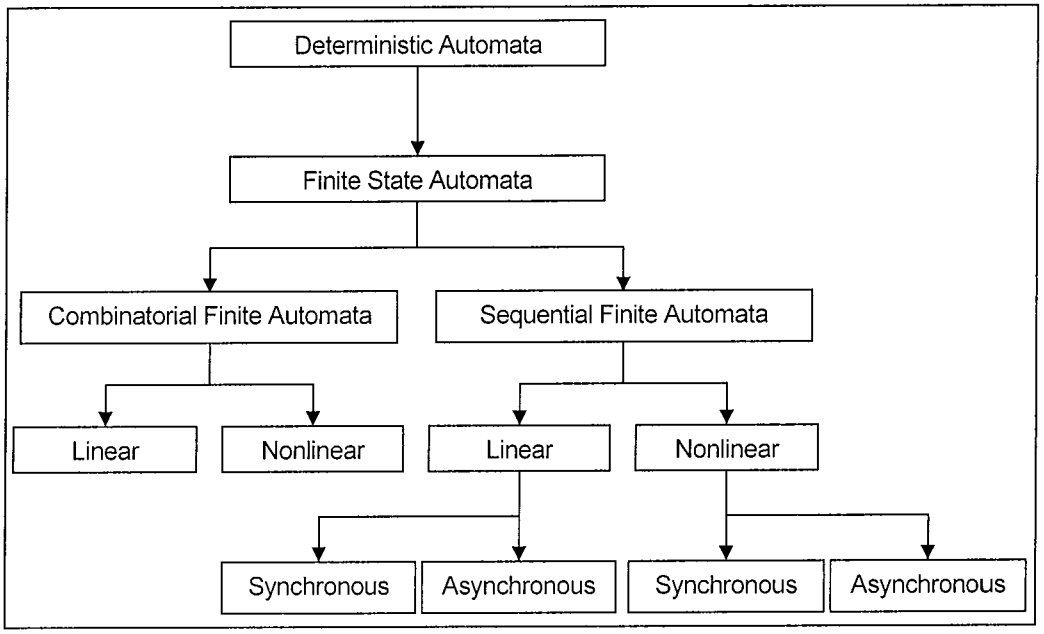

Figure 2.7: Classification of Automata

All subsystems described are used in CMOS integrated circuits. Deterministic automata are divided into finite state machines and further in combinatorial and sequential systems. Furthermore, the combinatorial and sequential finite state machines are subdivided into linear and nonlinear systems. The linear and nonlinear sequential finite state machines can be further divided into synchronous and asynchronous systems [Lunz06]. Additional, the general structure of a finite state machine can be recursive or non-recursive [Shep63]. The overall division is a particularly important feature for the analysis of integrated circuits because different automata types have different internal structures. Therefore, all possible structures must be considered for the analysis of finite state machines in unknown CMOS integrated circuits. In the following sections, all subgroups of finite state machines will be described with their internal structures, mathematical functions as well as the input and output parameters.

2.4.1 Combinatorial Finite State Machines

A characteristic of a combinatorial digital system is that at any given time the output values only depend on the particular combination of input values [Alma89]. Therefore, they are independent of input combinations, which were applied in the past. Combinatorial systems are called as static systems [Wuns93]. A combinatorial finite state machine has only one internal state. Since this internal state is not

changed, this information does not affect the processing of input values. Generally, the combinatorial finite state machine is described mathematically by (2.5).

$$A = (\underline{X}, \underline{Y}, f) \tag{2.5}$$

Such finite state machines do not have any internal storage and cannot buffer any sequential information [Lipp05]. This means that a combinational system has no possibility of storing information about previous inputs or to use such information to influence the output. The combinatorial finite state machine is described by the transformation F. The result function f maps the output values $\underline{Y}$ from the input values $\underline{X}$. Furthermore, the transformation F consists of the amount l of type f. The initial value is a transformation of the input values and is described by (2.6). The combinatorial finite state machine processes input values at discrete times. The same input values always lead to same output values [Alma89].

$$\underline{Y} = F(\underline{X}) \tag{2.6}$$

Combinatorial finite state machines have a very simple structure. They are based on three fundamental logic gates AND, OR and NOT [Stür71]. The NOT gate does not exist on its own and therefore, the AND and OR gates are linked with the NOT gate through NAND and NOR gates. In addition, a combinatorial network can be described with a multitude of small interconnected elementary automata. However, another representation is based on the AND and XOR operations. Together, these operations form the Galois field $GF(2)$ as defined in Section 2.4. Combinational finite state machines can be used to design devices to solve problems using simple logic. Here, logic operations such as addition, coding, error detection and multiplexer can be realised. Combinatorial finite state machines cannot store any information and therefore, they cannot execute any sequential algorithm. The input values are directly connected to the output without storing any value. Therefore, the output always depends solely on the combination of the input levels. Combinational finite state machines may contain an arbitrary number of logic gates but no feedback loops. A combinatorial finite state machine is shown in Figure 2.8.

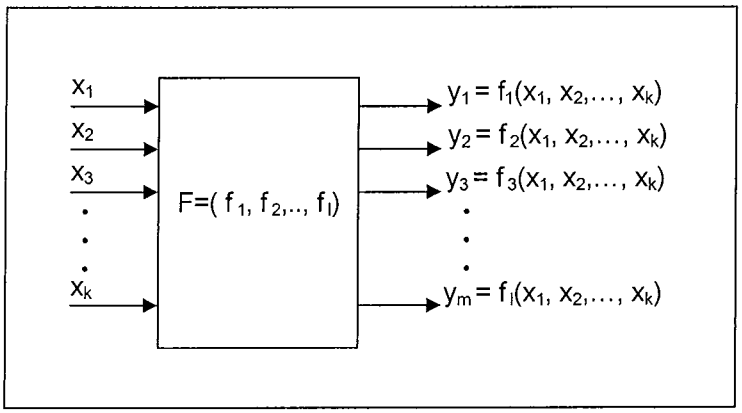

Figure 2.8: Combinatorial Finite State Machine

In this figure the input word $\underline{X}$ is represented by the vector components x_k whereas k is the number of inputs and $\underline{X}$ are the values of the input alphabet Σ. The output values are illustrated by the result functions f and depend solely on the input values. All result functions are combined in the output amount F. Combinatorial finite state machines are a subsection of deterministic automata as described in Section 2.3. They can be further subdivided into linear and nonlinear digital systems. In the next section sequential finite state machines are introduced.

2.4.2 Sequential Finite State Machines

Sequential finite state machines are a general category of finite state machines and are also called dynamic systems [Alma89]. For combinatorial digital systems the output is only a function of the current inputs applied, irrespective of any previous inputs. By contrast, using sequential digital finite state machines, the output can be a function of the currently applied input values as well as the sequence of previous input values. The previous input values can be stored and used later on. This is possible because sequential finite state machines have a memory where the different previous input values can be buffered. To form sequential finite state machines memory blocks are added to combinatorial finite state machines. This memory provides the ability to store values and to realise integrated system states. In general, the sequential finite state machines are classified into two different groups, acceptors and transducers. The acceptors work with the binary values *TRUE* or *FALSE*. Here, the output values are results of the internal states. These automata are used predominantly in language processing [Aste03]. The transducers generate the output values in dependence to the input values and the internal states. These properties of

finite state machines are used primarily to control procedures. The fundamental equation for the description of a sequential finite state machine is represented by a 6-tuple shown in (2.7).

$$A = (\underline{X}, \underline{Z}, \underline{Y}, z_0, g, f) \tag{2.7}$$

This 6-tuple is described by the input words $\underline{X}$, the internal states $\underline{Z}$ and the output words $\underline{Y}$ as previously introduced in Section 2.2. Additionally, sequential finite state machines have an initial state z_0. This state is particularly important for the determination of finite automata because of its possible influence on current or future output values. However, the transition function g characterises the internal transitions and is dependent on $\underline{X}$ and $\underline{Z}$. Every storage is described by one internal function. Consequently, the number of functions conforms to the number of storages. The number of output functions f corresponds to the number of outputs. The type of automaton determines whether the result function only depends on input values or on input values as well as the internal states [Stür71]. The input function and the result functions are presented in (2.8).

$$\begin{array}{l} g(\underline{X}, \underline{Z}) \\ f(\underline{X}, \underline{Z}) \end{array} \qquad or \qquad f(\underline{Z}) \tag{2.8}$$

The number of state elements indicates how many input values can be stored. The output values $y(t)$ are a function of input values $x(t)$ and of stored values $x(t\text{-}1), x(t\text{-}2), \ldots, x(t\text{-}n)$ or only a function of the stored values. Here, n is the number of system states. The input values applied $x(t\text{-}1), x(t\text{-}2), \ldots, x(t\text{-}n)$ correspond to the current internal states $\underline{Z}$. The transition function g depends on the input word $\underline{X}$ and on the internal vector $\underline{Z}$. Therefore, the number of functions which describe the sequential finite state machines can be different. The block diagram in Figure 2.9 shows the input parameters, the internal states and the output parameters of a sequential finite state machine.

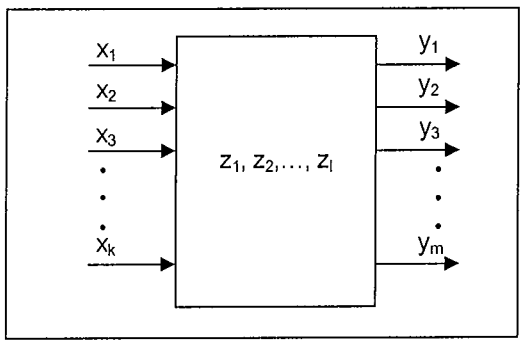

Figure 2.9: Sequential Finite State Machine

Sequential automata consist of an interconnection of combinatorial gates and storages. Furthermore, the internal structure can be different so that the functions $f(t)$ and $g(t)$ may depend on several variables. The general state and the result equations are presented in (2.9). The number of transition functions corresponds to the internal states. The number of result functions is always equal to the number of outputs.

$$\begin{aligned} \underline{z}_1(t+1) &= g_1\left(\underline{z}_1(t),\ldots,\underline{z}_l(t);\underline{x}_1(t),\ldots,\underline{x}_k(t)\right) \\ &\vdots \\ \underline{z}_l(t+1) &= g_l\left(\underline{z}_1(t),\ldots,\underline{z}_l(t);\underline{x}_1(t),\ldots,\underline{x}_k(t)\right) \end{aligned} \tag{2.9}$$

$$\begin{aligned} y_1(t) &= f_1\left(\underline{z}_1(t),\ldots,\underline{z}_l(t);\underline{x}_1(t),\ldots,\underline{x}_k(t)\right) \\ &\vdots \\ y_m(t) &= f_m\left(\underline{z}_1(t),\ldots,\underline{z}_l(t);\underline{x}_1(t),\ldots,\underline{x}_k(t)\right) \end{aligned}$$

Sequential finite state machines are further divided into linear and nonlinear systems and thereafter, it is possible to differ these two systems into synchronous and asynchronous automata as illustrated in Figure 2.7. The classification of synchronous and asynchronous systems may be further subdivided into recursive and non-recursive structures [Shep63]. In addition, the sequential finite state machines can be subdivided in accordance to the behaviour of the result function $f(t)$. Thus, the result function is a function of various parameters. These parameters determine the type and function of the automata type. The next section will give an overview of linear automata.

2.4.3 Linear Automata

In Mathematics, its applications in natural sciences and technology many tasks can be simplified if a linear relationship between values exists [Gill66], [Wuns93], [Oppe04], [Lunz06]. In system theory, and in particular in automata theory, an important special case occurs when a linear relationship between input, output and internal states exist. The automata representation of a dynamic digital system is characterised by a linear automaton if it has the fundamental form described in (2.10).

$$\begin{aligned} z(t+1) &= Az(t) + Bx(t) \\ y(t) &= Cz(t) + Dx(t) \end{aligned} \tag{2.10}$$

In these equations $x(t) \in \{0, 1\}^k$ are the k binary input signals, $y(t) \in \{0, 1\}^m$ are the m binary output signals and $z(t) \in \{0, 1\}^l$ are the l-dimensional states vector. The functions in (2.10) are linear transformations between finite input vectors, finite state vectors and finite output vectors. Furthermore, the elements A, B, C and D are matrices of suitable dimension with coefficients of the Galois field. The dimension of the matrices corresponds to the number of inputs, states and outputs. An automaton $A = (\underline{X}, \underline{Z}, \underline{Y}, z_0, g, f)$ is described as linear binary automaton if $\underline{X}$, $\underline{Z}$, $\underline{Y}$ are elements of a $GF(2) = \{0, 1\}$ and if the transition function and the result function have the following behaviour as shown in Equation (2.10). Linear finite automata consist of basic modules and are configured as linear sequential circuits. It can be seen from Equation (2.10) that these linear sequential circuits can be realised with additions, multiplications and delay gates only. Constant multipliers, shift registers and adders are used as basic modules. Figure 2.10 illustrates a constant multiplier realised by a conjunction function.

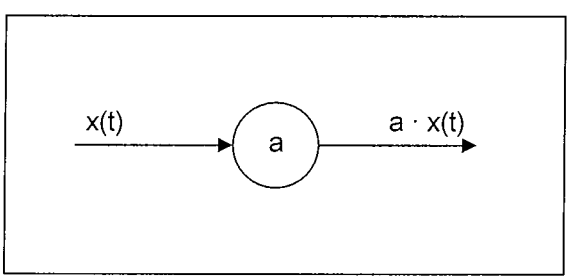

Figure 2.10: Constant Multiplier

The binary multiplication is considerably simpler than in the decimal system, since only multiplications by '0' or '1' occur. The result value $a \bullet x(t)$ of the multiplier a is calculated without a delay time t if the value $x(t)$ at t is applied to the input. In addition, linear sequential finite state machines work with $GF(2) = \{0, 1\}$ as described in Section 2.4. Therefore, the constant a of the multiplier can only be '0' or '1'. The result of the product formation corresponds to the AND gate in digital design.

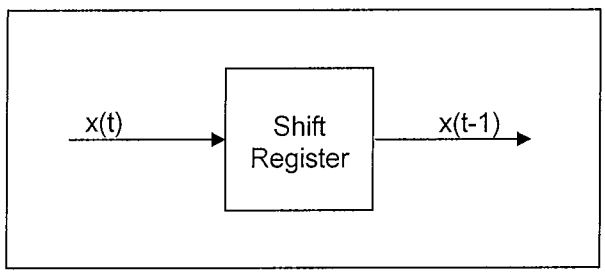

Figure 2.11: Shift Register

A shift register as a fundamental structure with input and output parameters is presented in Figure 2.11. A shift register does not cause any change of the input value, but the signal is delayed by one clock cycle. Furthermore, it represents essentially a data storage device with a storage capacity of one bit. These registers are described as memory. In automata theory serial and parallel structures of storages may be used. Bigger memories can be realised by different connection structures of multiple shift registers. In addition, a shift register can be used as recursive and non-recursive connections. For internal storage structures all shift registers are clocked simultaneously and the content of the shift register is denoted as a state. The internal state of the automaton is illustrated by the states of all registers. In most cases shift registers are realised using D-flip-flops.

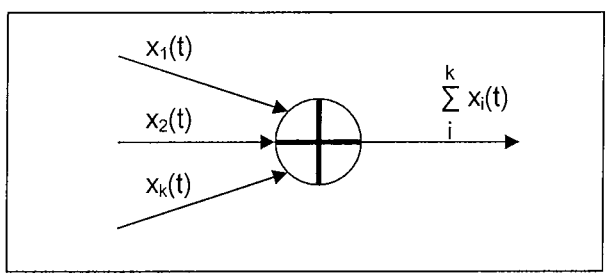

Figure 2.12: Adder for Digital Quantities

A binary adder with all input and output parameters is represented in Figure 2.12. An adder is realised in digital systems without a delay time. All input values $\underline{X}(t)$ are added and result in the sum $\Sigma X(t)$. The logical antivalence function corresponds to the addition modulo two. In the following section the division into synchronous and asynchronous finite automata is discussed.

2.4.4 Division into Synchronous and Asynchronous Finite Automata

Sequential finite state machines can be divided into synchronous and asynchronous digital systems as illustrated in Figure 2.7. Both types of systems are characterised by an external clock and both are used equally in integrated circuit design. Synchronous and asynchronous systems work differently each having advantages and disadvantages. Therefore, they will be considered separately in the following sections.

The characteristic of synchronous finite automata can be described as follows. All internal gates work within the same cycle time as represented in Figure 2.13. This means that they process their input signals after exactly defined points in time. But it is also possible that some gates have another divided clock signal. However, these divided clock signals work on the same clock edge. This clock controls the operation of all internal gates and furthermore, these gates are independent of all other parameters such as internal states, input and output values. Internal gates may only change their state during these cycle time points. This means, that the control signal determines whether a gate entity will respond or ignore transitions on certain input signals.

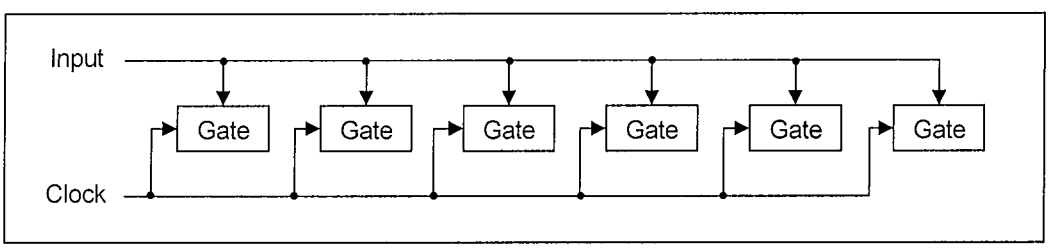

Figure 2.13: Example of a Synchronous Automaton

The propagation delay at synchronous finite state machines need not to be considered between the separate gates as the gates do not react to input signals of other gates. Through this construction, synchronous finite state machines are based

on a simple structure and because of this unstable states cannot occur. This means, that the system is stable between two clocked pulses. In this synchronous structure no interim values are calculated and used unless the clock signal is applied to the gates.

The fundamental design of asynchronous finite state machines is the same as that of synchronous finite state machines, but the allocation of the clock signal is different to that of synchronous finite state machines. The processing of values may be triggered either by a clock signal or by an input signal from another gate. Sequential systems are called asynchronous finite state machines if at least one internal gate does not process the input values in accordance with the clock edge.

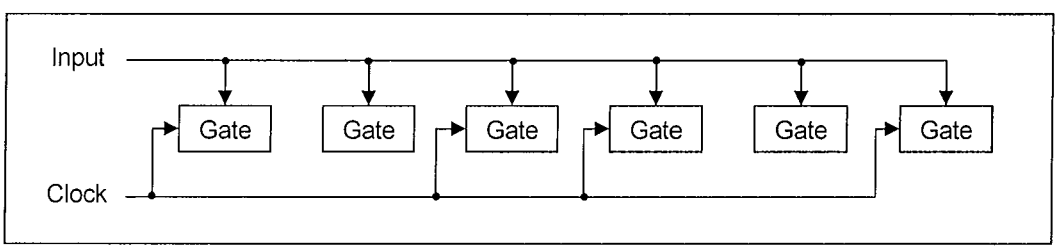

Figure 2.14: Example of an Asynchronous Automaton

In Figure 2.14 it can be seen that not all gates are connected to a clock signal. In the internal structure of asynchronous finite state machines gates are available which react to other parameters. These parameters are independent of clock signals. Because of this it is possible that gates work between the clock cycles. Variable signal paths in asynchronous finite state machines lead to different propagation delays in some gates and therefore, unstable internal states may occur [Wuns86]. In the next section different automata models are introduced which are used to describe the behaviour of the IC under investigation.

2.5 Automata Models

In automata theory different automata exist which describe the output behaviour in relation to several parameters. Two different fundamental sequential finite automata types are distinguished, transducers and acceptors. Acceptors recognise and accept the input values and the result is handed to the output by their internal state. This automaton type is used for example in speech recognition [Lunz06]. Furthermore,

transducers exist which are more complicated in their structures than acceptors. They generate output signals with respect to input values and internal states. Transducers are predominantly used in digital control systems [ibid]. These automata define a relationship between the output signals, the input signals and the internal state and form a sub class of sequential finite state machines. In addition, they are mathematical behavioural models with specific properties. By means of these automata different models can be described in mathematics, computer science and signal theory. Transducers are distinguished fundamentally between Mealy automaton [Meal55] and Moore automaton [Moor56]. These types of automata are particularly important for the description of abstract mathematical behavioural models in automata theory. The theory is very similar for both automata types. In this thesis Mealy and Moore will be introduced as abstract automata. These abstract automata will be described as mathematical structures. Their behaviour is illustrated with transition function and result function as in (2.8). Mealy and Moore automata are not mutually exclusive. A sequential finite state machine can be both a Mealy and a Moore automaton. This means, that an automaton may have some outputs depending on the inputs and the current state while some outputs may only depend on the current state. In addition, it is possible to convert a Moore structure into a Mealy structure and vice versa. Both automata models will be described in next two sections.

2.5.1 Mealy Automaton

The structure and function of a Mealy automat was originally developed by George H. Mealy and published in Bell System Technical Journal in 1955 [Meal55]. This publication describes the mathematical system, whereas the output signals are described in dependence on the internal states and the input values as can be seen in Figure 2.15. This means that two different possibilities exist. The output values depend on the change of the input values or on the change of the internal values.

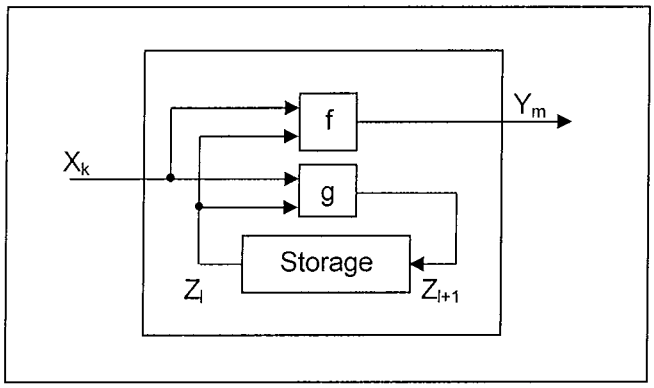

Figure 2.15: Structure of a Mealy Automaton

The fundamental behaviour is described by Equation (2.7). Furthermore, the structure of a Mealy automaton consists of the input vector $\underline{X}$, the state transition vector $\underline{Z}$ as well as the output vector $\underline{Y}$. However, Mealy machines are typically represented by the equations as shown in (2.11)

$$\begin{aligned} z(t+1) &= g\left(\underline{X}(t), \underline{Z}(t)\right) \\ y(t) &= f\left(\underline{X}(t), \underline{Z}(t)\right) \end{aligned} \tag{2.11}$$

where g is the transition function and f means the result function. With this specific model the internal behaviour depends on the input values and the state is determined. The Mealy automaton generates an output value depending on a change of the input value and the transition of a state into another state.

2.5.2 Moore Automaton

The Moore automaton also describes the principle behaviour of automata. In addition it can be shown that the Moore automaton also belongs to the sequential finite state machines. However, the result functions and internal structures are different from Mealy machine. Like the Mealy automaton the Moore automaton is a special case of the fundamental finite state machines. The internal structure and the function of the Moore automaton was originally developed by Edward F. Moore and published for the first time in the Princeton University Press 1956 [Moor56]. The Moore finite state machine also describes the mathematical behaviour of digital machines. Unlike the Mealy machine the output values $\underline{Y}$ and therefore, the output functions f of the Moore machine are independent of the input values $\underline{X}$. This means

that the output values are a function of the present states, but not of the present input values as can be seen in Figure 2.16.

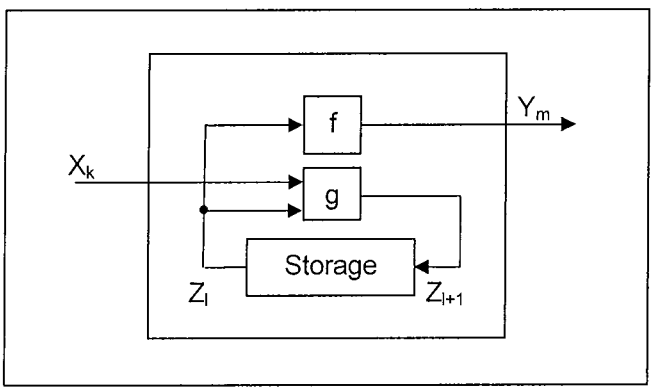

Figure 2.16: Structure of a Moore Automaton

The Moore machine also works with the input vector $\underline{X}$, the state transition vector $\underline{Z}$ and the output vector $\underline{Y}$. Different to the Mealy machine the result functions f are determined by the internal states $\underline{Z}$ only. The transition function g is illustrated by the input values $\underline{X}$ and the present states $\underline{Z}$ described by (2.12). This means, that the output can only change if the state changes too.

$$\begin{aligned} z(t+1) &= g\left(\underline{X}(t), \underline{Z}(t)\right) \\ y(t) &= f\left(\underline{Z}(t)\right) \end{aligned} \tag{2.12}$$

The fundamental mathematical structure is given by a 6-tupel as illustrated in (2.7). The transition function has the same structure as the general sequential finite state machine. Only the result function differs whether it is a Moore or Mealy automaton. In the next section the Medvedev automaton is explained.

2.5.3 Medvedev Automaton

Besides Mealy and Moore a third type of automaton the Medvedev automaton exist [Wuns86]. Furthermore, it is a simple form of automaton. The advantage of this automaton is the reduced amount of hardware as can be seen in Figure 2.17.

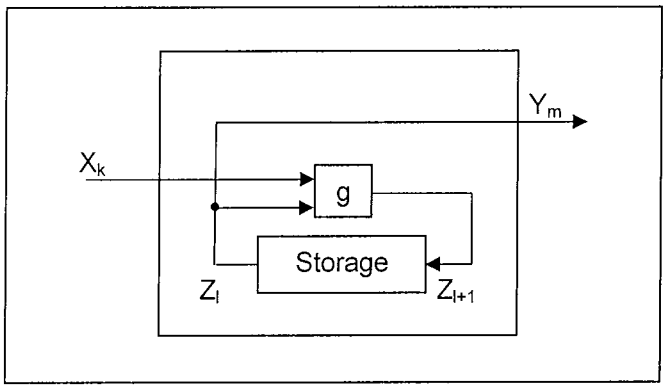

Figure 2.17: Structure of a Medvedev Automaton

The structure is a special case where the internal state itself is the output value. From the input signal and the present internal state the new internal state is calculated as described by Equation (2.13).

$$\begin{aligned} z(t+1) &= g\left(\underline{X}(t), \underline{Z}(t)\right) \\ y(t) &= \underline{Z}(t) \end{aligned} \tag{2.13}$$

The output of the Medvedev automata is the internal state itself. There is no separate output network with output function *f*. In the following sections a detailed overview of description forms of FSMs will be given.

2.6 Description Forms of Finite State Machines

The analysis of any given sequential circuit consists of the determination of chronological transition characteristic and the during this transition occurring output signals. Here, the transition characteristics result from the defined logical linking in the combinatorial circuit as well as from the applied input combinations. The analysis of FSMs is a particularly important part of automata theory because it is essential that the result of analysis must be represented in processable from. Hence, an automaton is characterised by its states and the state transitions between the states. With every clock signal the automaton changes into a new state or remains in the current state. During the state transition new values are provided as output parameters. However, it is possible to represent the functions of finite automata graphically, in tabular form or mathematically by matrices or by transition and result functions. These description forms are used to characterise synchronous automata. Here, it has been shown that the conversion from one presentation into another is

always possible [Balk93]. In the next sections the four description forms of FSMs are introduced.

2.6.1 State Diagram

State diagrams are used to describe the behaviour of sequential finite state machines in cases where their structure is not particularly complicated [Laws04]. The diagrams illustrate all possible states of an automaton if an event occurs and can be divided into Mealy and Moore structure. The state diagram is a very illustrative and expressive representation. Here, graphs consist of nodes and edges where the internal nodes are defined as states. Several nodes or states are illustrated by circles. The possible number of edges from node to another node depends on the number of input elements of the finite automaton. Since, the automaton has for example one binary input the number of output edges is equal to two. Usually, each state diagram represents objects of a particular model and tracks the different states of its objects through the system. A state diagram is called a directed graph diagram if a definite path can only be traversed in a certain way [Stür71]. State diagrams are used for many applications where it is necessary to understand or to demonstrate the behaviour of an entire system. In the following, the state diagrams of the Mealy and Moore the automaton will both be illustrated and explained using simple examples.

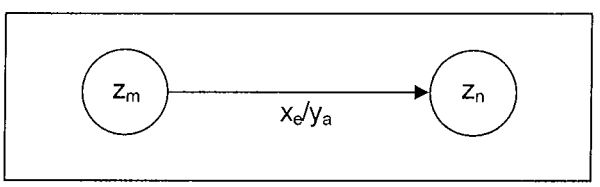

Figure 2.18: State Diagram of a Mealy Automaton

Figure 2.18 shows a simple state diagram of a Mealy automaton with its states, transitions and conditions. The property of a Mealy automaton is that the output only depends on the change of the input value or on the change of the internal state. Therefore, the output value y_a and the input value x_e are needed as input values for the next state, because both values y_a and x_e can change. The input value x_e is noted next to the corresponding edge that transfers the automaton from the state z_m into the state z_n. The resulting output value y_a is also written next to the corresponding edge. Inside the circles only the denotation of the internal states $\underline{Z}$ are included. In Moore

automaton the output behaviour is only determined by internal states. The output value y_p from the state z_m is the input value x_e for the next state z_n. The output element y_p is shown inside the circles as is illustrated in Figure 2.19.

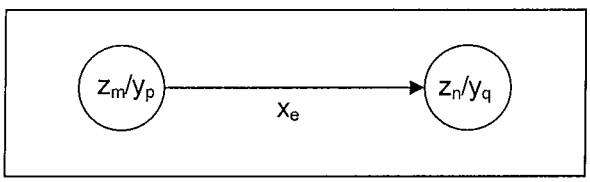

Figure 2.19: State Diagram of a Moore Automaton

The state diagrams of Mealy and Moore automata are different, but using state diagrams the internal sequences of both automata can be described. A state diagram is a good alternative to graphically explain and to illustrate the behaviour of automata. This representation is however, not practical for larger circuits and here other methods have to be used. In the next section another description form is described which is called state transition table.

2.6.2 State Transition Table

The chronological sequence of input variables, internal states and output variables can be described using a state table. Here, this table is a generalisation of a truth table which is used to analyse combinatorial systems. The difference is that the representation of the transitions between several states is essential in the state transition table. However, the output variables are a reaction to the system. In the state transition table the input and the state variables are available as independent input variables. This means that next to the 2^k input combinations 2^l different states are to be considered, so that the table has $2^k \bullet 2^l = 2^{k+l}$ rows. In this table the output variables $y_m(t)$, the excitation variables $e_n(t)$ and the state variables $z_l(t\text{-}1)$ of the result exist as independent variables. The excitation variables are the input signals of the delay or storage gates contained in the feedback. For the variables $e_n(t)$ and the output variables $y_m(t)$ the algebraic equations can be configured as a function of the input and state variables directly from the circuit as represented in (2.14).

$$\begin{aligned} \underline{Y}(t) &= f\left(\underline{X}(t), \underline{Z}(t)\right) \\ \underline{Z}(t+1) &= \underline{E}(t) = g\left(\underline{X}(t), \underline{Z}(t)\right) \end{aligned} \tag{2.14}$$

The binary values result from two equations of the column of the input vector and the output vector immediately by creating all combinations. A state transition table of an automaton defines all details from its state and it is possible to illustrate the complete mathematical behaviour. In the following the general form of a state transition table can be seen.

	x^n	z^n	e^n	z^{n+1}	y^n
	$x_1 \ldots x_k$	$z_1 \ldots z_l$	$e_1 \ldots e_r$	$z_1 \ldots z_l$	$y_1 \ldots y_m$
0	1	0	1	1	0
1	0	1	0	0	1
2	0	1	1	1	1
$2^{m+k}-1$	1	1	0	0	0

Table 2.1: Example of State Transition Table

Table 2.1 gives an overview of a general state transition table of finite state machines. The internal behaviour with input vector, internal states and output vector are described. This representation of automata is only suitable for small systems because the transition table will be particularly large when the automata become more extensive. In the following section the matrix is introduced which is another alternative to describe finite state machines.

2.6.3 Matrix

A matrix is another possibility to describe the behaviour of automata. In general, automata can be characterised by matrices which allow an analytic usage. Here, it is differentiated between adjacency matrices and incidence matrices [Kien06]. The quadratic $n \times n$ adjacency matrix $A(G)$ is shown in (2.15).

$$\underline{A}(G) = \left[a_{ij}\right] = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \tag{2.15}$$

Here, the matrix $\underline{A}(G)$ is assigned to the vectored graph of an automaton $G = [V, E]$ which consists of the number of states $V = v_1, \ldots, v_n$ and the number of state transitions E. The element a_{ij} represents the transition from the state i to the state j as in (2.16).

$$a_{ij} = \begin{cases} 1 & if \left[v_i, v_j\right] \in E \\ 0 & otherwise \end{cases} \qquad for\,(i, j = 1, \ldots, n) \tag{2.16}$$

If a transition from the state i to the state j exists then $a_{ij} = 1$. Otherwise $a_{ij} = 0$. After describing the adjacent matrix the incidence matrix will be explained. Again, $V = v_1, \ldots, v_n$ is the number of states and $E = e_1, \ldots, e_m$ is the number of state transitions of the graph considered. If the state transition e_l comes from state v_k or the next state is v_k it can be said that e_l is incident to v_k. The general structure of an incidence matrix $\underline{I}(G)$ is illustrated in (2.17).

$$\underline{I}(G) = \begin{bmatrix} i_{11} & i_{12} & \cdots & i_{1n} \\ i_{21} & i_{22} & \cdots & i_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ i_{n1} & i_{n2} & \cdots & i_{nm} \end{bmatrix} \tag{2.17}$$

$\underline{I}(G)$ of the graph $G = [V, E]$ is a $n \times m$ matrix with the elements i_{kl} as can be seen in (2.18).

$$i_{kl} = \begin{cases} 1 & if\ e_l\ is\ incident\ to\ v_k \\ 0 & otherwise \end{cases} \qquad \text{for}\,(k = 1, \ldots, n; l = 1, \ldots, m) \tag{2.18}$$

Using the incidence matrix the rows are related to the states and the columns are related to transitions. Thus, a state transition has to be entered twice in the matrix. First, the state transition is entered in the row of the previous state. Secondly, the state transition appears in the row of the next state. Any other elements of the column are equal to zero. In the next section the fourth possibility to describe FSMs will be given.

2.6.4 Mathematical Description

The mathematical behaviour of automata can be represented fundamentally with transition and result functions. These functions describe the internal structure using input and output parameters as well as internal states. Here, every internal state of the system is characterised by a transition function and every output is described using an output function. The system inputs and the internal states result in the parameters of the functions. However, the mathematical procedure of combinational

systems is described with the help of the switching algebra, which is a subset of the Boolean algebra. Boolean algebra was developed by a mathematician named George Boole and was published in 1854 [Bool54]. Moreover, it is described as a many-value domain. The switching algebra is described binary of {0, 1} and defines the elementary logic blocks of logic operations. Furthermore, this algebra provides the possibility to describe logic statements and to determine the truth of such statements [Dewe97]. The description occurs with three different logic operations, these are defined as AND, OR and NOT. The switching algebra expression is generated by tracing the flow of the input variables through the various logic operators. In addition, several postulates exist in the mathematical theory which form the fundamental set of laws of switching algebra. They are illustrated comprehensively in [Schu67]. These postulates are called axioms and are assumed as true. The fundamental postulates were developed originally by Edward V. Huntington and published in 1904 [Hunt04]. Furthermore, switching algebra is used for the analysis of a collection of logic operations connected together to form a combinational digital system. This kind of description is not very concise and vivid, but it is possible to illustrate particularly large automata structures.

2.7 Summary

In this chapter a comprehensive discussion of the theoretical background of automata theory was carried out which will be used in the next chapters to describe unknown ICs to be investigated. After providing an overview of synthesis and analysis the model of finite state machines was described in Section 2.2. In the next sections the general classification of deterministic automata in combinatorial, sequential finite state machines and their subgroups was described. In addition, the internal structures, functions and the mathematical behaviour of these subgroups were explained. Section 2.5 then introduced different automata models which are used to describe the behaviour of the unknown IC to be investigated. To mathematically describe FSMs an overview of possible description forms was given in Section 2.6. However, from this chapter it can be concluded that it is possible to abstract unknown integrated circuits using automata theory. In the next chapters the conventions introduced in this chapter will be used to determine the function of an unknown integrated circuit.

3 Analysis of Different Finite State Machines

In the previous chapter the automata theory was discussed which can be used for an abstraction of an unknown integrated circuit. This chapter will now explain the analysis process to investigate unknown ICs. At first, Section 3.1 will give a general overview of the analysis procedure developed which consists of three main parts. In Section 3.2 the determination of pin types and their location will be explained which is the first step of the overall analysis procedure. This knowledge is important to further analyse the unknown IC. Using the information of the pins the next step is the separation procedure which will be introduced in Section 3.3. The classification procedure consists of three main parts. Here, the analysis starts with the determination of independent blocks to avoid misinterpretation of the overall IC. After describing the determination of independent blocks the classification in several types of FSMs will be given. Depending on the behaviour found different solution types will be presented. If the procedure results in combinatorial or linear sequential FSMs the respective identification procedures will be explained in Section 3.3.3 or Section 3.3.5. If non-recursive linear behaviour was found by the separation procedure Section 3.3.6 then presents a novel method to determinate the number of state number using the cross-correlation. In the case of investigated nonlinear behaviour Chapter 4 will extensively describe the identification process.

3.1 General Overview

This section describes the overall structure of the novel non-invasive analysis procedure which is obtained only by investigating the IC's input-output behaviour. In this process, the determination of pin types, the preliminary separation and the identification of the unknown IC are the three main parts of the analysis procedure. Figure 3.1 shows the analysis flow in general.

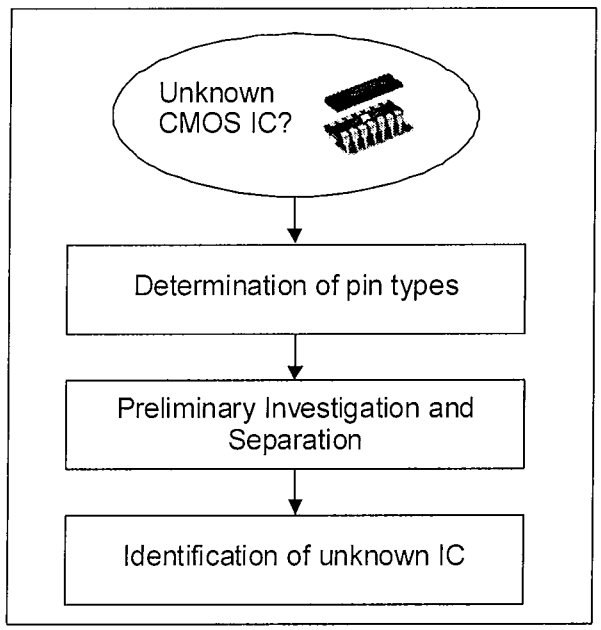

Figure 3.1: The Analysis Flow

First, several properties of the unknown integrated circuit under investigation must be identified. Here, the determination of pin types provides the number of available pins, the pin types and their locations on the unknown IC. With this information it is possible to further analyse the unknown device. In the second part of the analysis procedure the IC to be investigated is decomposed into parts of same behaviour to simplify further investigation. Here, the models of finite state machines described in Chapter 2 are used to abstract the unknown integrated circuit. Using the classification of automata, which was also explained in Chapter 2, a division into combinatorial, linear and nonlinear sequential FSMs is possible. The division of unknown ICs to be investigated is followed by the identification. Here, the identification of such IC is the third and most complex part of the analysis procedure. In the case of combinatorial or linear behaviour the identification will be explained in Section 3.3. However, in the case of nonlinear behaviour of the unknown IC Chapter 4 describes the further division into Moore or Mealy automata. After that, the analysis of the FSMs is carried out. The next section will introduce the determination of pin types.

3.2 Determination of Pin Types

For an optimised procedure it is necessary to use all available information of the IC to be analysed. In this case the number of pins and the package type are the only

information given. The first step of the analysis cycle is the determination of pin types and their location which is now described using the integrated circuit 74 HCT 00 as a simple example. As can be seen in Figure 3.2, the device contains four independent NAND gates [Phil09]. In the example used, the IC has a standard plastic Dual-In-Line package (DIP). Furthermore, the number of pins is 14.

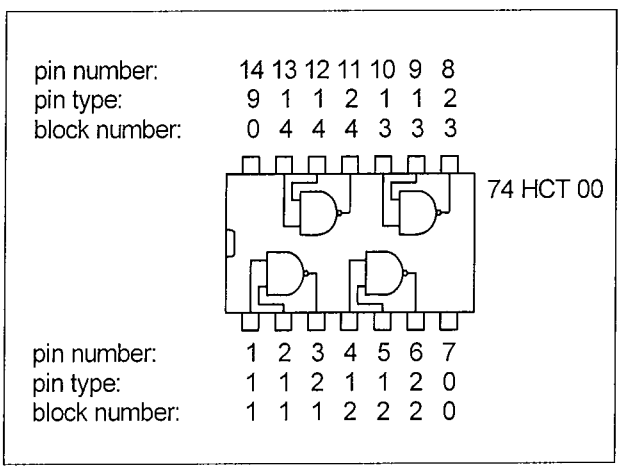

Figure 3.2: The Connection Diagram of the IC 74 HCT 00

When analysing unknown digital CMOS ICs it is important to focus on the different types of input-output relationships which can occur. The basic pin types might be of the type input, output or bidirectional. Furthermore, a tristate pin might be present. Some integrated circuits do not use all pins for electrical connection. In this case it is possible that one or more pins are not connected at all. Additionally, voltage supply pins and ground pins exist. For a fast and uniform process it is important to define a value for the different pin types which may occur in CMOS technology. The notations in Table 3.1 are used to describe the pin types in general, where assignment number eight is reserved for future applications. The organisation with numbers was chosen to allow easier processing using automated software tools.

Pin Type	Assignment
Ground	0
Input	1
Output	2
Bidirectional	3
Tristate	4
Clock	5
Reset	6
Not connected	7
Supply Voltage	9

Table 3.1: Notations to Describe the Input-Output Structure

Sequential ICs may also have clock or reset pins. Therefore, they are also a part of possible input pins. Due to the abstraction of unknown ICs to automata theory it is important to know the location of the clock and reset pins. Therefore, they have their own assignment numbers.

The determination of pin types is based on the electrostatic discharge (ESD) phenomenon described in [Grea92], [Dabr98], [Amer02], [Ker05]. The ESD phenomenon originates from the transfer of electrostatic charges between two objects with different electrical potentials. This results in damage to integrated circuits due to large energy dissipation in an extremely short time of less than 150ns. The ESD specifications of commercial IC products are generally higher than 2kV in human-body-model (HMB) ESD stress and 200V in machine-model (MM) ESD stress. To provide the ESD protection for CMOS ICs against unexpected ESD damages in the internal circuits of CMOS ICs, the on-chip ESD protection circuits have to be designed and placed around the input, output and power pads. However, the protection is important to clamp the overstress voltage across the internal circuits. Hence, it is important to provide a low impedance path to ground to discharge the ESD current of several amperes. The locations of the ESD protection circuits to achieve whole-chip ESD protection for CMOS ICs are illustrated in Figure 3.3, where the measurement paths are marked with arrows and numbers.

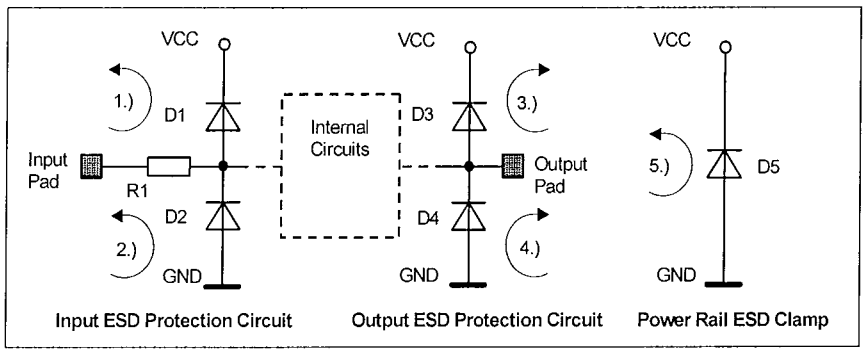

Figure 3.3: Typical Design of on-chip ESD Protection in CMOS ICs

It can be seen in Figure 3.3 that there is no difference in the electrical behaviour between the paths Output→VCC (3.), GND→Output (4.) and GND→VCC (5.) in theory. The power rail ESD clamp (5.) with the diode D5 has always the lowest threshold voltage in the ESD protection circuits compared to paths (3.) and (4.) with the diodes D3 or D4, respectively. This phenomenon can be explained with another type of doping of the diode D5. The measurement paths Input→VCC (1.) and GND→Input (2.) always have a resistor R1 in series connection. Therefore, the slope of the input ESD protection circuits is lower than the slope of the output ESD protection circuits and the power rail ESD clamp. The overall ESD protection circuit characteristics are shown on the basis of the example IC 74 HCT 00 in Figure 3.4.

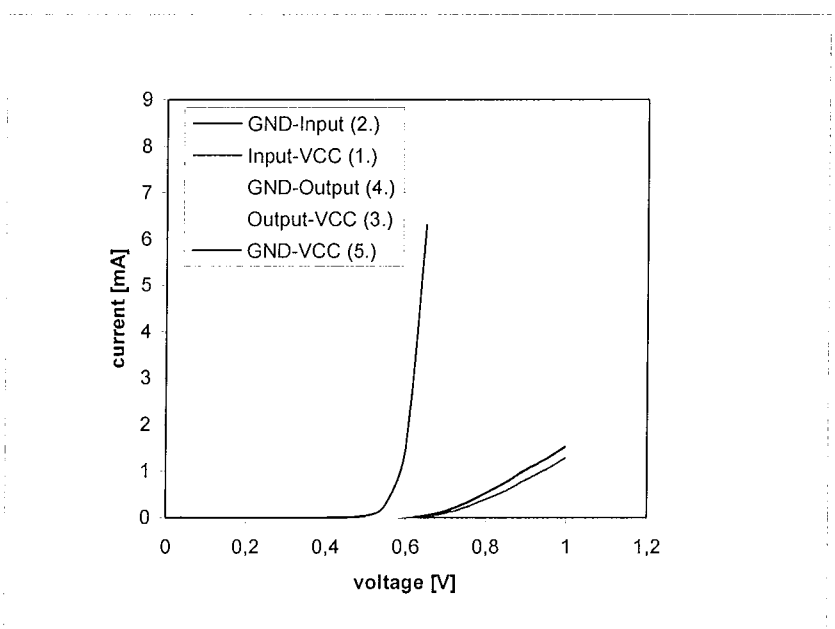

Figure 3.4: The *i-v*-Characteristics of ESD Protection Circuits (IC 74 HCT 00)

As can be seen in Figure 3.4 that the *i-v*-characteristic leads to a voltage measurement reading of 1mA. Therefore, a current source of 1mA is be used. To avoid damages in the IC a maximum supply voltage of 3,3V is used. Figure 3.5 shows the measurement setup in general. The measurement will be carried out using the measurement setup shown in Figure 3.5.

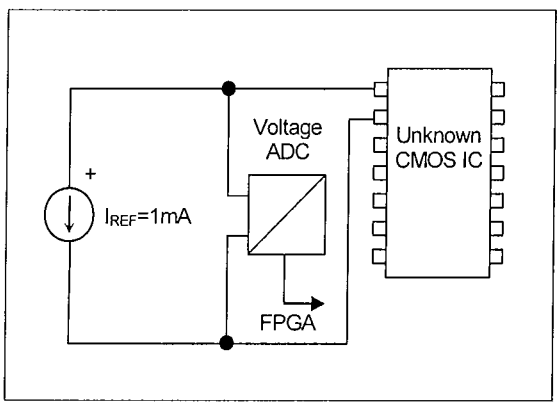

Figure 3.5: The Overall Measurement Setup

The number of measurement readings m can be written as follows

$$\begin{aligned} m &= (n-1)\cdot n \\ m &= (14-1)\cdot 14 \\ m &= 182 \end{aligned} \tag{3.1}$$

where n is the number of pins of the unknown CMOS IC. In the example described 182 different voltage values have to be measured with all possible pin-to-pin combinations. First, the measured values have to be converted from analogue to digital values. Next, the values will be sent to the FPGA running the evaluation software. Here, the data is stored in the data table. After the measurement is completed the evaluation is started. The path ground to supply voltage in the example IC 74 HCT 00 is identified between pin 7 and pin 14 as described in (3.2).

$$path_{GND-VCC} = Min(v) = path_{7-14} \tag{3.2}$$

The result can be checked using (3.3):

$$path_{VCC-GND} = Max(v) = path_{14-7} \tag{3.3}$$

After the identification of pin types VCC and GND the truth table can be reduced significantly. Only voltage values measured from pin 7 (GND) to the other pins have to be considered. The remaining values can be deleted. In the new data table the following pin types can be determined as follows:

$$\begin{aligned} &pin_{n.c.} = Max(v) \\ &pin_{Input} < pin_{n.c.} \; AND \; pin_{Input} > pin_{Output} \\ &pin_{Output} < pin_{Input} \end{aligned} \tag{3.4}$$

The results are stored in the allocation table which is illustrated in Table 3.2. Furthermore, Table 3.2 shows the complete input-output structure in the standardised form for the example IC 74 HCT 00. It can be seen that the example circuit consists of eight inputs labelled as '1' and four outputs labelled as '2'. Hence, the voltage supply pin ('9') and the ground pin ('0') are written into the table. This information is needed in the next step of the procedure.

Pin number	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Pin type	1	1	2	1	1	2	0	2	1	1	2	1	1	9

Table 3.2: Example for the Input-Output Structure of the IC 74 HCT 00

Now, the information about the location of the voltage supply pin, the ground pin and the types of pins in relation to the pin number are known. The following section will show the separation procedure to simplify the analysis of unknown ICs.

3.3 Separation Procedure

The determination of pin types is followed by the classification procedure to minimise the following analysis of the IC. As described in Chapter 2 automata can be generally divided into deterministic and non-deterministic automata. In this thesis, only deterministic finite state machines A will be considered, which are defined by (3.5).

$$A = (\underline{X}, \underline{Z}, z_0, \underline{Y}, g, f) \tag{3.5}$$

Again, $\underline{Z}$ is the set of state symbols: $\underline{Z} = \{z_1, z_2, \ldots, z_r\}$, r is the number of state symbols, $z_0 \in \underline{Z}$ is the initial state, $\underline{X}$ is the set of input symbols: $\underline{X} = \{x_1, x_2, \ldots, x_m\}$, m is the number of input symbols, $\underline{Y}$ is the set of output symbols: $\underline{Y} = \{y_1, y_2, \ldots, y_l\}$, and l is the number of output symbols. It is assumed that the FSM A operates at unit time intervals. Letting $x(t) \in \underline{X}$, $y(t) \in \underline{Y}$ and $z(t) \in \underline{Z}$ be the input symbol, output symbol and state symbol at time t, respectively, then the FSM A is described by the discrete dynamical system in the form of

$$A\begin{cases} z(t+1) = g(z(t), x(t)) \\ y(t) = f(z(t), x(t)) \end{cases} \tag{3.6}$$

A further division into different types of automata is possible. Finite state machines can be subdivided into combinatorial, linear sequential or nonlinear sequential automata as presented in Chapter 2. The separation procedure is used making the following realistic assumptions.

Assumption 1: Every single pin, such as input pins, output pins, voltage supply pins etc. are already determined. Therefore, the number of input symbols $\underline{X}$ and output symbols $\underline{Y}$ are known. The determination of pin types was already described in detail in Section 3.2. Due to clarity, this section will only consider input and output pin types, even so the determination of pin types presented in the previous section is able to cope with all relevant pin types.

Assumption 2: The maximum number of state symbols r is known. Furthermore, the FSM investigated has a synchronous clock line and reset capability to set the FSM into the initial state z_0 after each simulation run.

Assumption 3: The state transition function g and output function f of the IC are unknown.

In principle the separation procedure works in three steps. The determination of independent blocks, the division into combinatorial or sequential FSMs and the division into linear or nonlinear sequential FSMs can be seen in Figure 3.6.

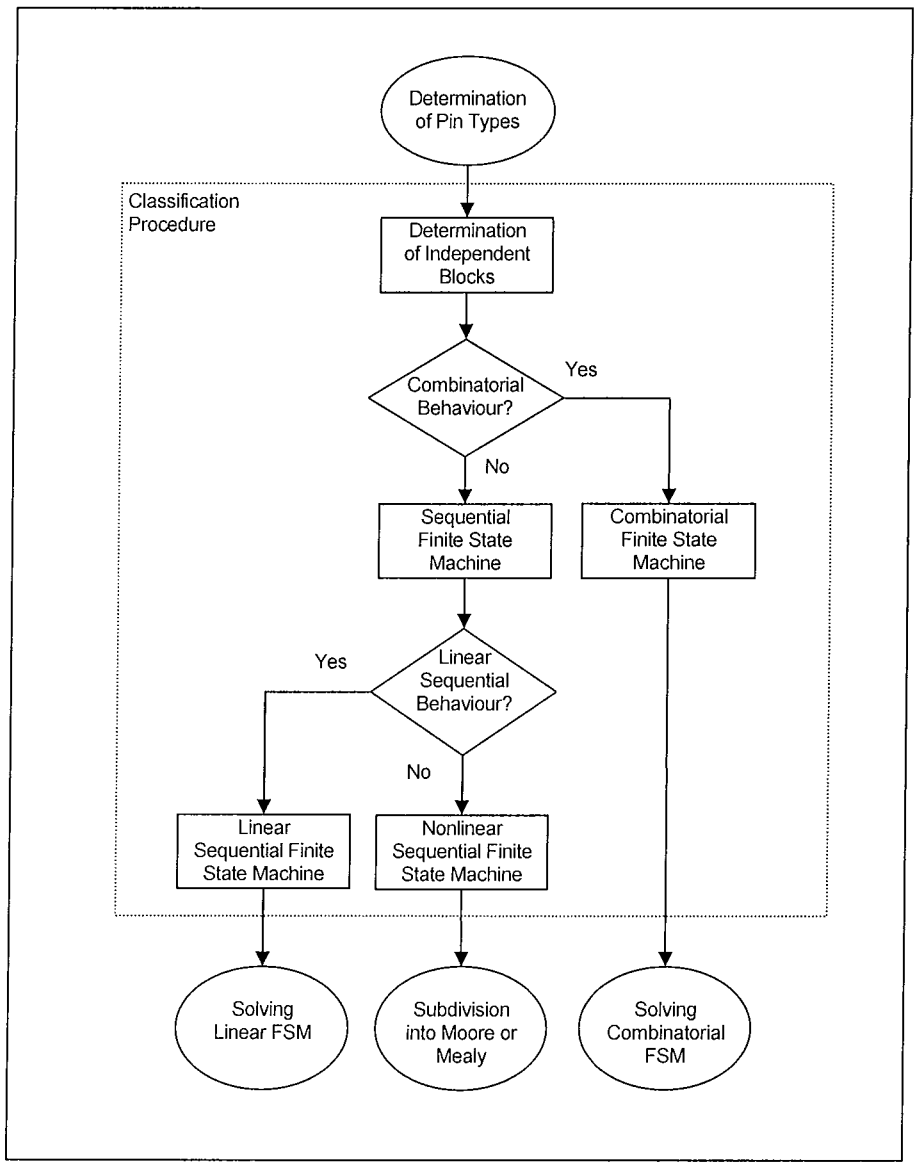

Figure 3.6: The Classification Procedure of Finite State Machines

First, the unknown integrated circuit is divided into independent blocks, where every block has its own behaviour. Furthermore, every independent block consists of its inputs and outputs. Afterwards, every single block is checked for combinatorial behaviour. If the block has a combinatorial behaviour the separation procedure is complete and no further classification analysis of the block has to be done. Here, for every combinatorial block all 2^i bit combinations have to be applied, where i is the number of inputs of the combinatorial block. Then the output function can easily be determined and minimised. Otherwise, the block investigated must have a sequential behaviour and then a further division into linear or nonlinear sequential behaviour is carried out. This is done by checking the block for linear sequential behaviour. First

several abstract IC models were designed using MATLAB [Matl08]. They contain all types of automata to demonstrate the mode of operation as represented in Figure 3.6. After having shown the overall classification procedure an example of an IC model can be seen in Figure 3.7. The example consists of three different independent blocks, eight input pins and seven output pins.

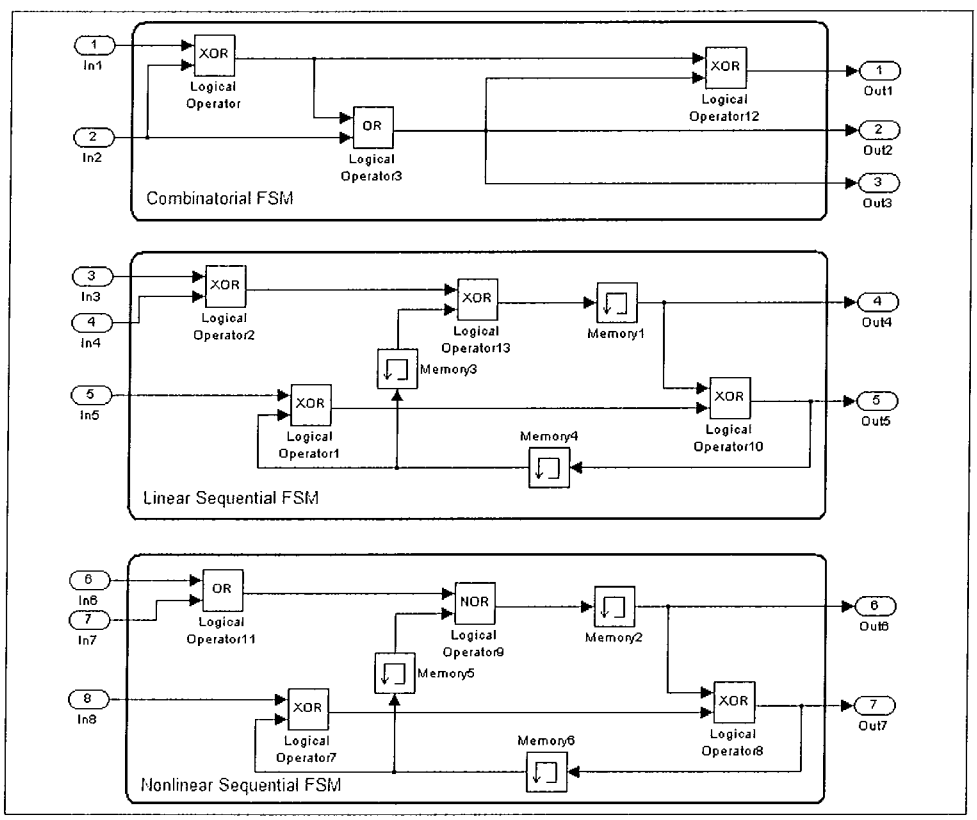

Figure 3.7: Example of a Virtual IC Model Implemented into MATLAB

The first block consists of the inputs x_1 and x_2 as well as the outputs y_1, y_2 and y_3. It is designed using simple logic gates and for this reasons it is a combinatorial finite state machine. The second block contains the input pins x_3, x_4 and x_5 and the output pins y_4 and y_5. Furthermore, it consists of linear logic gates as well as additional memories. Therefore, this block is a recursive linear sequential system. The third block contains the inputs x_6, x_7 and x_8 as well as the outputs y_6 and y_7. Here, instead of a linear XOR gate a nonlinear OR gate is used and thus, the design is a recursive nonlinear sequential automaton. Figure 3.7 acts as an example to demonstrate the operation and can be enlarged or modified. It should be noted that a real IC input can be linked to multiple blocks, where the blocks can be of the different automata types such as combinatorial, linear or nonlinear sequential. The next section describes the

determination of independent blocks which is the first part of the separation procedure.

3.3.1 Determination of Independent Blocks

Since the unknown integrated circuits usually consist of a very complex structure they can be composed of several independent blocks with different characteristics. The interdependencies between the internal blocks are analysed first to avoid a false interpretation of the whole IC. Moreover, the determination of independent blocks is important for the separation of the unknown system, because the complexity of the following analysis procedures can be reduced significantly. Figure 3.8 illustrates the basic structure of an unknown integrated circuit with its different finite state machines. Here, the unknown system consists of j independent blocks, i input pins and w output pins. Every independent block is defined by its input and output pins for example block j consists of $(n+1)$-i inputs and $(y+1)$-w outputs.

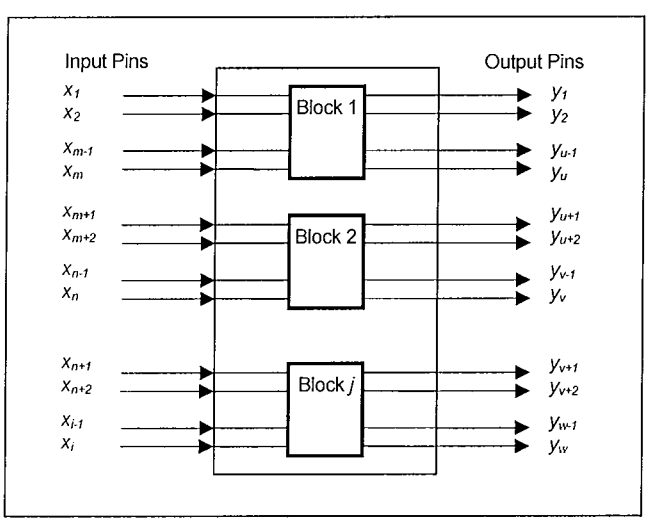

Figure 3.8: Basic Structure of a Multidimensional Recursive Finite State Machine

Each independent block can have a different behaviour. For example two automata can simultaneously exist within one IC. With every clock cycle one sequential automaton has one possible active state. Since, finite state machines are defined by $GF(2)$ two distinct states are possible. Furthermore, finite state machines exist with several concurrent states, which are independent from each other. This automata structure is called parallel automaton. A parallel automaton might have multiple active state variables. The events which describe the state transitions cannot occur

simultaneously but are independent of each other. The determination of the behaviour of parallel finite state machines is particularly difficult because interdependences must be considered since multiple internal states can occur, which elapse in parallel at the same time. Therefore, the number of possible state combinations is much larger than the total number of internal states. A parallel structure with two processes is illustrated in Figure 3.9. Here, two active states are available and the combinations of these states must be considered.

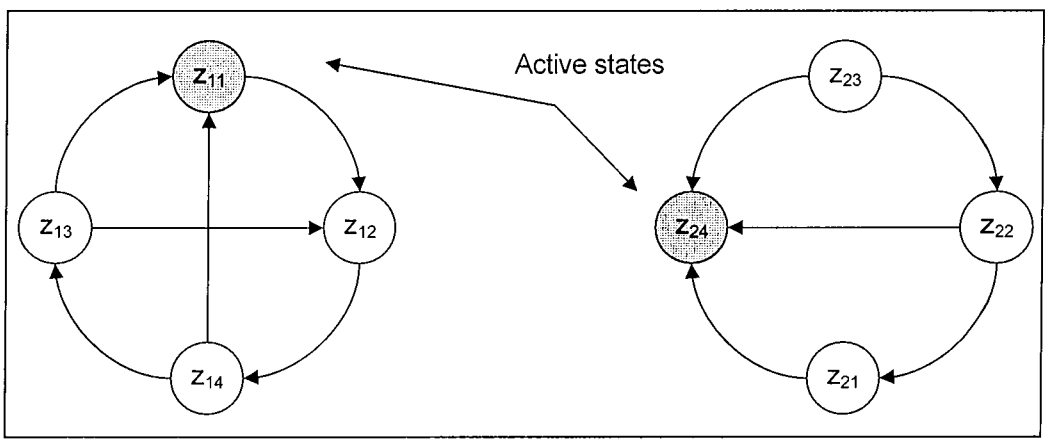

Figure 3.9: Example of Parallel Automata

These different automata structures are only one example of possible configurations. Many more other finite state machines models exist, which show a different mathematical behaviour. Therefore, the next analysis step must deal with the investigation of all possible structures in integrated circuits. Generally, the determination of independent blocks is carried out in three steps. First, different combinations of test sequences are applied at the input, then the output values have to be stored and finally, equations for the interdependencies must be solved.

In this research the test vectors must ensure a high switching activity inside the unknown IC. However, it is only possible to determine the right properties of an IC under investigation if the output reacts as often as possible. Due to the fact that only binary deterministic automata are considered in the classification procedure each sequence must consist of uniformly distributed values '0' and '1'. Furthermore, the generators for such test vectors have to be easily implemented. To satisfy these requirements the linear maximum sequences were chosen. They are an important subclass of pseudo random sequences [Fing97]. However, they are used in signal processing in many applications to fulfil a given task [ibid]. The theoretical

derivation of linear maximum sequences will be described in the following. After that the autocorrelation function (ACF) is introduced to prove the uniform distribution of values '0' and '1'. The practical realisation of such sequences will be explained afterwards. Therefore, linear feedback shift registers (LFSRs) are explained which are used in the research.

In general, linear homogeneous finite difference equations can be written as (3.7).

$$c_0 a_i + c_1 a_{i-1} + \ldots + c_n a_{i-n} = 0, \qquad i = n, n+1, \ldots, \qquad c_0 c_n \neq 0 \tag{3.7}$$

Using Equation (3.7) an infinite sequence of elements as in (3.8) can be defined.

$$\{a_i\}_0^{\infty} = a_0, a_1, a_2, \ldots \tag{3.8}$$

Rewriting (3.7) these sequences are recursive, because each element a_i can be calculated from n previous elements as shown in (3.9).

$$a_i = -\frac{1}{c_0} \sum_{v=1}^{n} c_v a_{i-v}, \quad c_v, a_i \in GF(q) \tag{3.9}$$

The elements of the sequences and the recursive coefficients will be taken from a Galois field. A formal power series as in (3.10) can be assigned to a sequence $\{a_i\}_0^{\infty}$.

$$G(D) = a_0 + a_1 D + a_2 D^2 + \ldots + a_i D^i + \ldots \tag{3.10}$$

Equation (3.10) is also called a generating function or D-transformed of the sequence [Pete67]. Linear recursive sequences, which are generated by (3.7) and (3.9) are always periodic. By using (3.9) it can be seen in (3.11) that the particular initial condition according to Equation (3.11) only repeats itself and has the period $N = 1$.

$$(a_{-n}, a_{-n+1}, \ldots, a_{-1}) = (0, 0, \ldots, 0) \tag{3.11}$$

Consequently, the maximum period length is illustrated in (3.12) because there are q^n-1 different n-digit vectors $s \neq 0$ with elements from $GF(q)$.

$$N = N_{\max} = q^n - 1 \tag{3.12}$$

A linear recursive sequence which satisfies (3.7) and which furthermore, has a period length as in (3.12), can be called q-significant maximum sequence of the order n. Therefore, the period length N of a maximum sequence is independent of its initial condition. Moreover, all maximum sequences which satisfy one and only one finite difference equation only differ in the shifting of its values. These two properties result in many advantages when analysing an unknown IC for example no specific initial conditions are necessary except $s \neq 0$ due to practical realisations of these sequences. Short disturbances do not influence the generated sequence. For this, the maximum sequence will be generated from the set of linear recursive sequences. Using the periodicity as shown in (3.10) the equation can be rewritten as in (3.13):

$$G(D) = [1 + D^N + D^{2N} + \ldots] \sum_{i=0}^{N-1} a_i D^i \tag{3.13}$$

Using the identity as described in (3.14)

$$1 = \left(1 - D^N\right)\left(1 + D^N + D^{2N} + \ldots\right) \tag{3.14}$$

the special binary case can be written as follows:

$$\frac{1 - D^N}{c(D)} = \sum_{i=0}^{N-1} a_i D^i \tag{3.15}$$

If the exponent e is equal to the period length N then it follows from (3.15) that the characteristic polynomial $c(x)$ is the divider of binomial $(1\text{-}D^e)$. In the case of the maximum period all initial conditions $s \neq 0$ are equitable. This means that a special initial state represents no constraint. Furthermore, an essential condition for the maximum period length is a characteristic polynomial. The minimum number e for which $(1\text{-}x^e)$ can be divided without a remainder is called exponent or period of the polynomial $f(x)$. Therefore, a polynomial with a maximum exponent $e = N_{max}$ must be irreducible. A polynomial $f(x)$ of the order n can be called primitive if $(x^N\text{-}1)$ can be divided by $f(x)$ without a remainder, with $N = q^n\text{-}1$. However, this is not valid for $N < q^n\text{-}1$.

After theoretically deriving the linear maximum sequences, a function is now needed to prove the uniform distribution of values '0' and '1'. Here, the determination of the autocorrelation function (ACF) of maximum sequences provides sufficient information about this distribution [Fing97]. Therefore, the ACF is introduced in the following. The special characteristic of the ACF depends on the choice of amplitude values A_i assigned to its elements of $GF(q)$. Due to the use of binary maximum sequences the autocorrelation function with the assignment $A_0 = -1, A_1 = +1, x_i \in \{A_0, A_1\}$ results in (3.16).

$$R_{x,x}(s) = \begin{cases} 1 & s \equiv 0 \bmod N \\ -\dfrac{1}{2^n - 1} & otherwise \end{cases} \tag{3.16}$$

Figure 3.10 illustrates Equation (3.16). Here, it can be seen that the ACF is constant for all values $s = 0 \bmod N$. Furthermore, the higher the orders of n the faster the autocorrelation function falls towards zero.

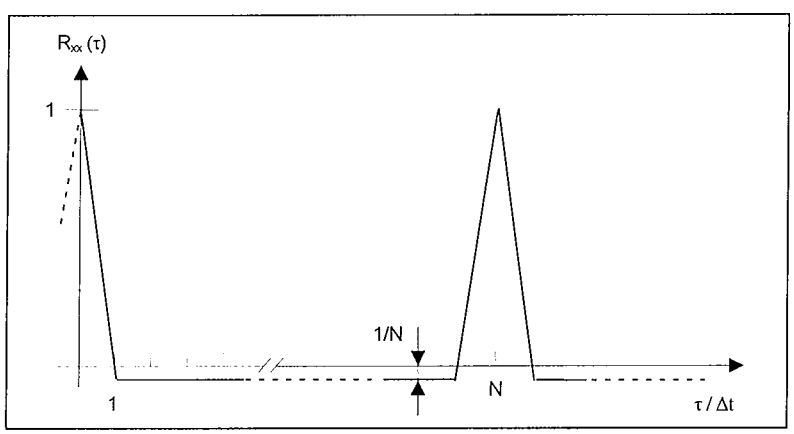

Figure 3.10: ACF of Binary m-Sequences

Referring to the example in Figure 3.7 the number of input pin is equal to ten. Therefore, for each input pin of the unknown IC a different sequence has to be generated to avoid the permanent application of same input vectors. Thus, ten different test sequences have to be generated which satisfy the properties previously described. In this research generator polynomials which lead to the binary maximum sequences of the length 2^{10}-1 = 1023. Table 3.3 shows the characteristics of binary maximum sequences investigated and with their generator polynomials and initial states used in this research.

Number of pn-sequence	Generator Polynomial	Initial State
1	[1 0 0 1 0 0 0 0 0 0 1]	[0 0 0 0 0 0 0 0 0 1]
2	[1 1 0 1 1 0 0 0 0 0 1]	[0 0 0 0 0 0 0 1 0 0]
3	[1 1 1 0 0 1 0 0 0 0 1]	[0 0 0 0 0 0 0 1 1 1]
4	[1 0 1 1 0 1 0 0 0 0 1]	[0 0 0 0 0 0 1 0 1 0]
5	[1 0 1 0 0 1 1 0 0 0 1]	[0 0 0 0 0 0 1 1 0 1]
6	[1 1 1 1 0 1 1 0 0 0 1]	[0 0 0 0 0 1 0 0 0 0]
7	[1 0 0 0 0 0 0 1 0 0 1]	[0 0 0 0 0 1 0 0 1 1]
8	[1 1 0 1 0 0 0 1 0 0 1]	[0 0 0 0 0 1 0 1 1 0]
9	[1 0 1 0 0 0 1 1 0 0 1]	[0 0 0 0 0 1 1 0 0 1]
10	[1 1 1 0 1 0 1 1 0 0 1]	[0 0 0 0 0 1 1 1 0 1]

Table 3.3: Generator Polynomials for pn-sequence Generation

The autocorrelation functions of all generator polynomials shown in Table 3.3 were determined and successfully checked that their ACF leads to maximum at (0). This is the proof that each pn-sequence has a uniform distribution of values '0' and '1'. Furthermore, at each input of the IC under investigation a different binary maximum sequence has to be applied to avoid the same input pattern on two or more inputs at the same time. All autocorrelation functions were determined using the program IDL. Figure 3.11 shows the result of the ACFs which are all equal and satisfy the requirements.

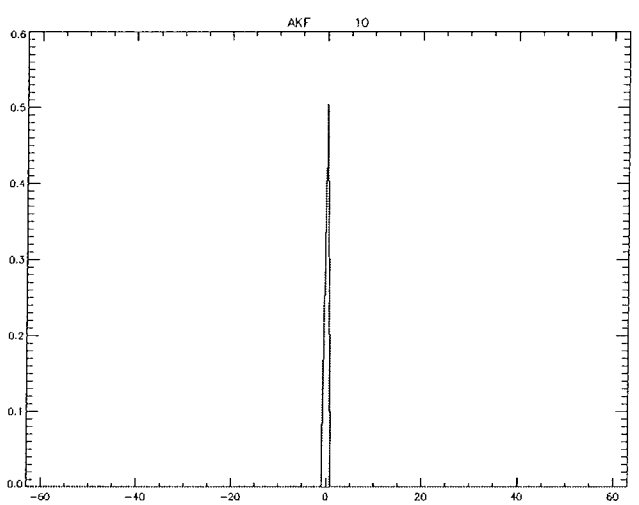

Figure 3.11: ACF of Used Binary m-sequences

Now, the implementation of the test vectors investigated is described. For the generation of test vectors linear feedback shift registers are used. They are easy to implement and can be seen in the following Figure 3.12 [Matl08].

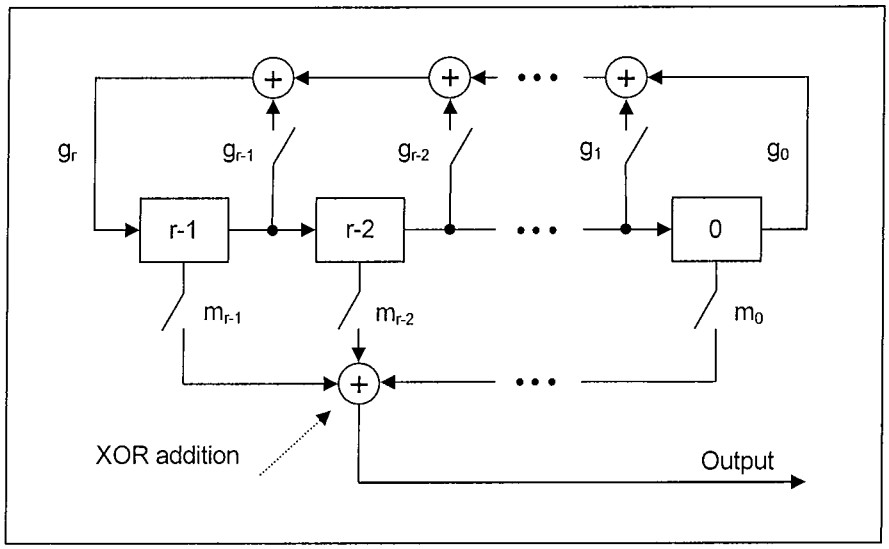

Figure 3.12: LFSR for Generation of Binary m-sequences

In Figure 3.12 it can be seen that LFSRs use a simple shift register generator configuration. At each time step the adders perform an addition modulo 2 according to the incoming arrow to the shift register If there is a connection from the k^{th} register to the adder the coefficient $g_k = 1$. The shift register is described by the primitive binary generator polynomial as in (3.17).

$$p(z) = g_r z^r + g_{r-1} z^{r-1} + g_{r-2} z^{r-2} + \ldots + g_0 \tag{3.17}$$

Referring to the first row in Table 3.3 the pn-sequence pn_1 generator polynomial $p_1(z)$ can be written as in (3.18).

$$pn_1 = p_1(z) = z^{10} + z^7 + 1 \tag{3.18}$$

In the following, referring to the example in Figure 3.7, i different pn-sequences have to be applied to the inputs in 2^i-1 different combinations. Here, i is the number of input pins as illustrated in Table 3.4.

Number of Combination	Inputs x_i							
	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8
1	pn_1	0	0	0	0	0	0	0
2	0	pn_2	0	0	0	0	0	0
3	pn_1	pn_2	0	0	0	0	0	0
4	0	0	pn_3	0	0	0	0	0
...	...	...	...	...	...	...	...	...
253	pn_1	pn_2	pn_3	pn_4	pn_5	pn_6	0	pn_8
254	pn_1	pn_2	pn_3	pn_4	pn_5	pn_6	pn_7	0
255 (2^1-1)	pn_1	pn_2	pn_3	pn_4	pn_5	pn_6	pn_7	pn_8

Table 3.4: Example of a Data Table Generated

Next, for every pn-sequence combination the results of each output have to be recorded and stored in a data table. Here, for Table 3.4, 255 output sequences have to be recorded. Finally, the dependencies can be solved by forming equations for each output. If an output reacts to a particular combination of pn-sequences applied then the output depends on the appropriate inputs. For example, if an output responds to the 254^{th} combination represented in Table 3.4, it depends on the inputs x_2-x_8. If the resulting sequence of the same output with the 255^{th} combination applied is equal to the former combination, the output depends on every input except the last one. Each output, which has the same input dependencies, belongs to the same internal block. For the example described in Figure 3.7 the resulting array is given in Table 3.5.

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8
y_1	1	1	0	0	0	0	0	0
y_2	1	1	0	0	0	0	0	0
y_3	1	1	0	0	0	0	0	0
y_4	0	0	1	1	1	0	0	0
y_5	0	0	1	1	1	0	0	0
y_6	0	0	0	0	0	1	1	1
y_7	0	0	0	0	0	1	1	1

Table 3.5: Result Array of Independent Blocks

In the table above it can be seen, that three different independent blocks exist. Equation (3.19) shows the mathematical interpretation of Table 3.5.

$$\begin{aligned} B_1 &= f(y_1, y_2, y_3, x_1, x_2) \\ B_2 &= f(y_4, y_5, x_3, x_4, x_5) \\ B_3 &= f(y_6, y_7, x_6, x_7, x_8) \end{aligned} \tag{3.19}$$

Using the described separation procedure three independent blocks were found. In the first block B_1 inputs x_1 and x_2 are interconnected with outputs y_1, y_2 and y_3. Circuit block B_2 is connected between inputs x_3, x_4 and x_5, and outputs y_4 and y_5. Internal circuit B_3 is connected between inputs x_6, x_7 and x_8 and outputs y_6 and y_7. The result extracted from the separation procedure is equivalent to the example of the integrated circuit from Figure 3.7. After the determination of the independent blocks the division into combinatorial or sequential FSMs will be presented in the next section.

3.3.2 Division into Combinatorial or Sequential Finite State Machines

In current microelectronics many different types of integrated circuits exist and these may be described with the help of deterministic finite automata structures. The division in automata types will simplify the analysis because some characteristical features of the internal behaviour will be identified. In addition, it will be shown that combinatorial and sequential structures are the main discriminating factor of finite state machines and consequently of integrated circuits. In Figure 3.13 the fundamental analysis procedure is illustrated and with this classification of unknown systems the further analytic process can be determined.

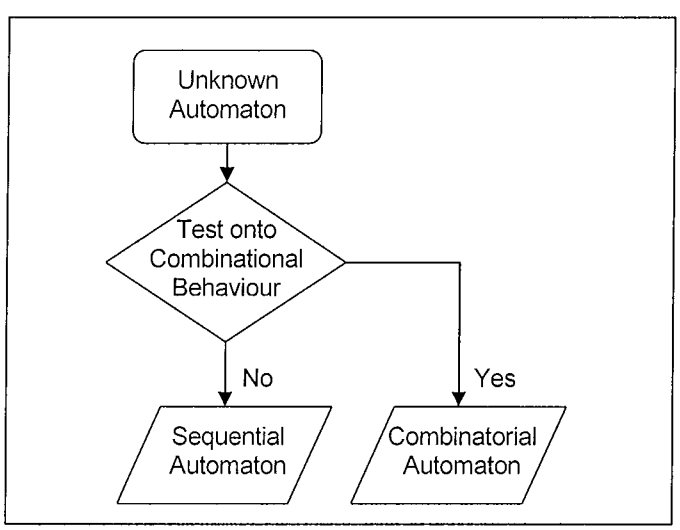

Figure 3.13: Flowchart of Deterministic Automata Classification

The classification into combinational and sequential systems is the next step of the analysis process of unknown integrated circuits. This determination is particularly important because the subsequent analysis steps depend on this classification. This analysis does not describe the internal structure but rather classifies the system into sequential or combinatorial behaviour. The division is possible because both systems have different internal structures. The fundamental behaviour model may be tested using several input combinations because the input values of both systems are processed differently. This means that the combinatorial and sequential structures respond differently to the input combinations applied. The test is carried out by assigning certain input combinations and recording the corresponding output combinations. Since the internal structures vary, the reactions to the input combinations will be different. In combinational systems no internal storage exists. This means that the input parameters are directly linked to the outputs. Therefore, the system outputs respond immediately and always in the same manner to the input values. Consequently, the output values are associated with the input values applied and the same input combination must always generate the same output combination. Furthermore, all possible input combinations must be applied in a different order, ensuring that an influence of previous combinations is avoided. Sequential systems have internal storage and may also have feedback loops. Therefore, the output behaviour of sequential structures in relation to the input combination is different from combinatorial systems. The output values of sequential automata may depend on the current input combination, the previous input combination and the internal states. Therefore, the output values behave differently than the combinatorial automata. By means of this behaviour the output values can be separated into sequential and combinatorial structures. With this first test the basic structure of the investigated system can be determined. In the following, tests of certain subgroups will be executed to determine further properties and structures.

The same pn-sequences as described in the previous section are used for the division into combinatorial or sequential finite state machines. This time, rather then combinations of pn-sequences, many completely different pn-sequences are applied to all inputs of one independent block at a time. The pn-sequences have to be different every time because identical input patterns must be avoided. This is

because various possible internal states have to be considered, otherwise the analysis may lead to false results. In the second step the results of the output pins as well as the values of the input pins of the corresponding time steps are stored in a data table. An example with four inputs and two outputs is shown in Table 3.6.

Time Index t	**Inputs x_i**				**Outputs y_k**	
	x_1	x_2	x_3	x_4	y_1	y_2
1	1	0	1	0	0	0
2	0	1	0	0	1	0
3	**1**	**1**	**0**	**1**	**0**	**1**
...						
254	1	1	0	0	1	
255	**1**	**1**	**0**	**1**	**0**	**1**
256	0	1	1	0	1	1
...						
510	1	0	0	1	0	1
511	**1**	**1**	**0**	**1**	**0**	**1**
512	0	1	0	0	1	0
...						

Table 3.6: Example of a Data Table Recorded

Using the results in Table 3.6 for each time step *t* the results are checked using the theory of combinatorial logic [Koha78]. According to this theory the same input values must always lead to the same output results. In the example lines 3, 255 and 511 have the same input combination and therefore, must lead to the same output results for a combinatorial system. This analysis can easily be carried out by using the same input patterns for every time step. If the results show that the system is a combinatorial finite state machine the classification procedure is complete and the system can be analysed further as will be described in the next section. On the other hand, if only one identical input combination does not lead to the same output result the check can be stopped immediately because the system is a sequential finite state machine. Then, a further subdivision into linear or nonlinear sequential automata is necessary as will be described in Section 3.3.4.

3.3.3 Analysis of Combinational Finite State Machines

If the device under investigation is determined to be a combinatorial system, then the further investigation of the unknown combinatorial IC is executed. The analysis process only deals with the description of the mathematical transition functions. This

process is divided into theoretical simulation using MATLAB and the illustration of the results in a truth table, where the system inputs and outputs are provided. In addition, the transition functions are determined by means of the truth tables. The number of transition functions corresponds to the number of system outputs. The internal circuit of combinational systems can be composed of many logic elements. Due to many possible structures the internal circuit is ignored during the analysis. The same mathematical functions can be realised using different structures so that the logic elements used and internal connections are not needed for analysis [Boch75].

Combinatorial structures are the simplest types of deterministic finite state machines. The internal circuits only consist of fundamental logic gates. These combinational circuits have no internal storage. Consequently, the input pins are directly linked to the output pins without storage. Therefore, the output values are functions of the current input combination [Wuns93]. This property simplifies the determination of the transition function. The characterisation of combinatorial FSMs may occur using the information about the sequence of input combinations and the associated output values. All possible input combinations must be applied and the corresponding output values are stored. Hence, the same input combination must always lead to the same output combination. This property will be used during the analysis as well as during the determination of internal functions. In Figure 3.14 a simple combinatorial structure is shown which was implemented into MATLAB to demonstrate the correct operation.

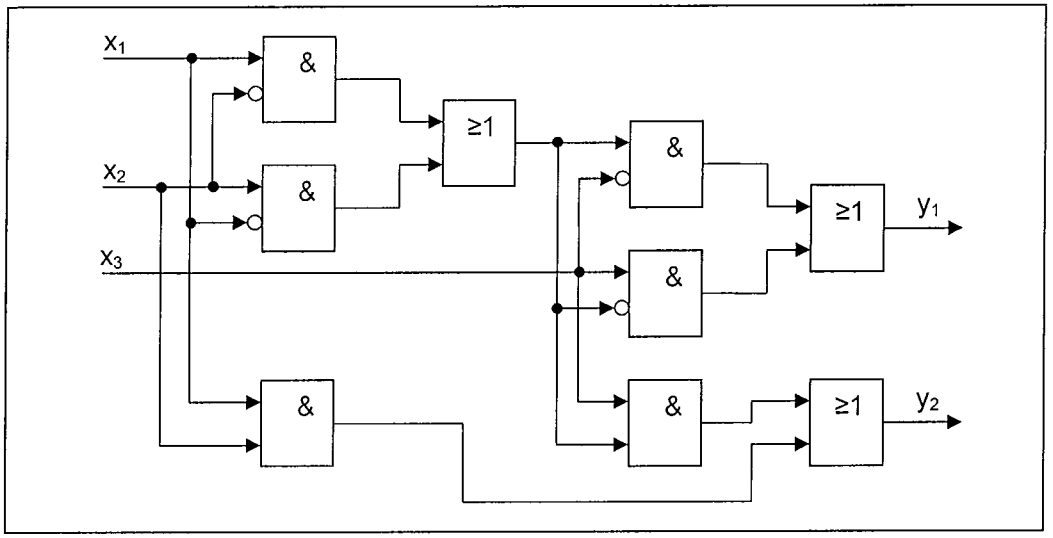

Figure 3.14: Example of a Combinatorial Digital System Design

The automaton in Figure 3.14 shows the typical characteristic features of combinatorial systems and will demonstrate how such a system is solved. The functional behaviour of this system is described as a full adder and consists of several logic gates. Furthermore, in the example described only the inputs x_1, x_2 and x_3 as well as the outputs y_1 and y_2 are known. Thus, the analysis only uses the three system inputs and two system outputs for the determination of the mathematical behaviour.

The test starts with the application of all possible input value combinations. Here, the corresponding output values will be recorded in a truth table. The resulting output values y_1 and y_2 define the input to output behaviour of the combinational digital system for all input combinations. For the analysis of the example in Figure 3.14 three inputs are used. Therefore, $2^3 = 8$ input combinations are required. Table 3.7 shows the result for eight possible combinations of three input variables.

x_1	x_2	x_3	y_2	y_1
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Table 3.7: Truth Table of Combinational Digital System

Logic expressions are useful to illustrate the analysis results. Each system output combination is generated by propagating the input variables. The analysed system is unknown and different internal logic elements can be used. Therefore, the direct propagation of the applied input combination cannot be determined. The combinational digital system is analysed and the system output combinations are listed in a truth table. These results are used for the description of the whole combinatorial system. In this section the identification of unknown combinatorial FSMs was carried out. The next section will explain the further division if the IC investigated has sequential behaviour.

3.3.4 Division of Sequential Automata into Linear and Nonlinear Systems

If the unknown finite automaton shows a sequential internal behaviour as was described in Section 3.3.2 then the investigated automaton can be further divided into linear and nonlinear structures. In this section this classification and the analysis of linear sequential structures will be described. This subdivision simplifies the analysis of unknown automata because the unknown structures are subdivided using basic structures. Then the analysis will be started and the exact mathematical behaviour will be examined. The classification into linear and nonlinear behaviour is illustrated in Figure 3.13 and is executed with the help of a process of elimination. This means that the unknown finite state machines are tested in relation to their linear properties.

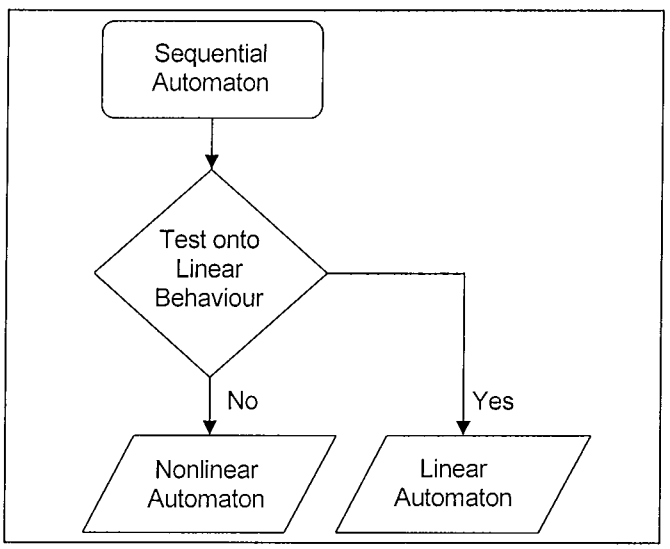

Figure 3.15: Flowchart of Linear and Nonlinear Classification

Basically, this separation is based on the linear superposition of linear automata. The linear superposition is one of the theoretical most interesting properties of linear automata as described in [Wuns86] and [Oppe04]. Again, it is important to consider Equation (2.10) as a description of linear automata. This equation can be also written as shown in (3.20) and (3.21):

$$\begin{aligned} z_i(t+1) &= \sum_{j=1}^{n} \alpha_{ij} z_j(t) + \sum_{j=1}^{q} \beta_{ij} x_i(t) \\ &= f_i\left(z_1(t), \ldots, z_n(t); x_1(t), \ldots, x_q(t)\right) \end{aligned} \tag{3.20}$$

$$\begin{aligned} y_i(t) &= \sum_{j=1}^{n} \gamma_{ij} z_j(t) + \sum_{j=1}^{q} \delta_{ij} x_j(t) \\ &= g_i\left(z_1(t), \ldots, z_n(t); x_1(t), \ldots, x_q(t)\right) \end{aligned} \tag{3.21}$$

As can be seen from (3.20) and (3.21) the main basic circuit elements of a linear automaton are additions, time shifts and constant multipliers. These properties will be used for the classification of unknown systems into linear and nonlinear automata. In (3.22) $x(t)$ is the input symbol, $y(t)$ is the output symbol and $z(t)$ is the state during the clock cycle t, $z(t+1)$ is the state during the clock cycle $t+1$.

$$x(t) = \begin{pmatrix} x_1(t) \\ \vdots \\ x_q(t) \end{pmatrix}, \quad y(t) = \begin{pmatrix} y_1(t) \\ \vdots \\ y_m(t) \end{pmatrix}, \quad z(t) = \begin{pmatrix} z_1(t) \\ \vdots \\ z_n(t) \end{pmatrix} \tag{3.22}$$

Equation (3.23) shows the constant matrices A, B, C and D, whose elements of the field $K = GF(p)$. In this research the prime number $p = 2$, because only binary cases are considered.

$$\begin{aligned} A &= \begin{pmatrix} \alpha_{11} & \cdots & \alpha_{1n} \\ \vdots & & \vdots \\ \alpha_{n1} & \cdots & \alpha_{nn} \end{pmatrix} \quad B = \begin{pmatrix} \beta_{11} & \cdots & \beta_{1q} \\ \vdots & & \vdots \\ \beta_{n1} & \cdots & \beta_{nq} \end{pmatrix} \\ C &= \begin{pmatrix} \gamma_{11} & \cdots & \gamma_{1n} \\ \vdots & & \vdots \\ \gamma_{m1} & \cdots & \gamma_{mn} \end{pmatrix} \quad D = \begin{pmatrix} \delta_{11} & \cdots & \delta_{1q} \\ \vdots & & \vdots \\ \delta_{m1} & \cdots & \delta_{mq} \end{pmatrix} \end{aligned} \tag{3.23}$$

From the general block diagram as in [Wuns93] and Equations (3.20) and (3.21) the following property can be directly derived. If an automaton only consists of linear

basic elements, the whole automaton must be linear. Furthermore, it is possible to extend a linear automaton using linear elements. The whole automaton will be linear. If only one element has nonlinear behaviour the automaton is nonlinear.

In signal and system theory the one-dimensional step function is traditionally used to determine the transition function of a one-dimensional linear system as shown in [ibid]. Here, two problems have to be solved. First, the structure of ICs might be of adverse structures. This means for example, that if the circuit has a large number of AND-combinations the classification cannot be solved using the step function. Therefore, the division is also carried out using pn-sequences to increase the internal switching activity of the IC investigated. Here, the same sequences as for the determination of independent blocks can be used. However, many real integrated circuits have more than one input and also more than one output. In this case the pn-sequences have to be adapted to multi-dimensional applications. First, the pn-sequence has to be applied to the first input of one particular sequential block, while the other input pins are assigned with zeros. With every step the output values have to be recorded to create the corresponding matrix. For example, if the sequential block consists of three inputs then the first pn-sequence pn_1 has to be applied to the first input, while inputs two and three are assigned zeros and the output values are recorded. Then, pn-sequence pn_2 has to be applied to the second input, while the other two input pins are assigned zeros and the output values are recorded. Finally, the same is done for the third input using pn-sequence pn_3. Afterwards, the sum function has to be composed of the recorded output values. Therefore, the binary values for each output result where only one pn-sequence is applied to the input are added without considering the carry again. Next, all pn-sequences are applied to the inputs at the same time and the output results are recorded again. Finally, the system is analysed for linear sequential behaviour. This is done by comparing the results of the sum function with the result function, where all pn-sequences are applied at the same time. If the sums are equal the output has a linear behaviour, otherwise it has a nonlinear behaviour. This analysis has to be carried out for each output of a particular sequential block. Only if the results for the analysis of every output have shown to be linear, the whole block is a linear

sequential finite state machine, otherwise the system must be a nonlinear sequential automaton.

Many tests with different IC models have confirmed that the determination of sequential linear behaviour is possible using a test procedure proposed by [Reus69]. This classification procedure was implemented into MATLAB and various unknown sequential finite state machines were investigated towards their linear behaviour. Next, an example is given and on the basis of this linear finite automaton the identification of linear finite state machines will be explained. A linear system is represented by the transition function (3.24) and the output function (3.25).

$$z(t+1) = \underbrace{\begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}}_{A} z(t) + \underbrace{\begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 1 \end{pmatrix}}_{B} x(t) \tag{3.24}$$

Due to the matrices A and B in (2.10) the following states of a linear automaton are defined in relation to the values of input and the present states. In addition, the exact relation between the states and between the states and the inputs are defined. The number of internal states and inputs is given by the dimensions of the matrices.

$$y(t) = \underbrace{\begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \end{pmatrix}}_{C} z(t) + \underbrace{\begin{pmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix}}_{D} x(t) \tag{3.25}$$

The output functions are described on the basis of the present internal states and the combinations of input values. Matrix C describes the internal states and the matrix D characterises the different input values. The equations of linear systems describe the exact implementation of linear structures. The input vectors are k-dimensional, the state vectors are l-dimensional and the output vectors are m-dimensional. Therefore, the linear automaton has k inputs, l states and m outputs. On the basis of the matrix dimensions the relations between the automata elements are exactly determined. Here, it must be considered that linear finite state machines consist of several elements as described in Section 2.4.3. In Figure 3.16 the implementation of the linear sequential FSM is illustrated. Using this linear structure the classification and the subsequent analytic process will be shown.

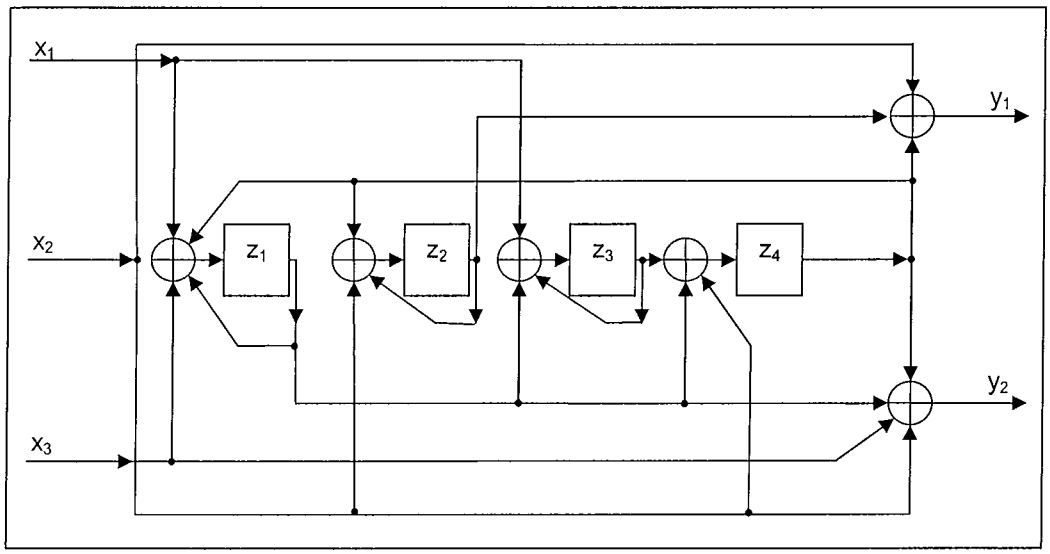

Figure 3.16: Example of a Sequential Linear Automaton

The features of a linear system are the elements used for the internal construction which is given by (3.24) and (3.25). This linear automaton was implemented into MATLAB and has $k = 3$ inputs, $l = 4$ internal states and $m = 2$ outputs. On the basis of this example the linear identification and the subsequent investigation in relation to the internal function will be represented. The classification of unknown systems in relation to the linear or nonlinear behaviour is carried out with the help of two different analysis procedures [Wuns86]. This is possible because linear and nonlinear systems show different mathematical behaviour if a Dirac function is applied at the inputs [Boch81]. The analysis procedures use multiple pn-sequences applied to the unknown system. For each test cycle the impulse responses of each output are investigated. The classification is carried out by analysing multiple impulse responses. In this process two different measurements are carried out to analyse the unknown finite state machine with the help of impulse responses. This procedure is shown in Figure 3.17.

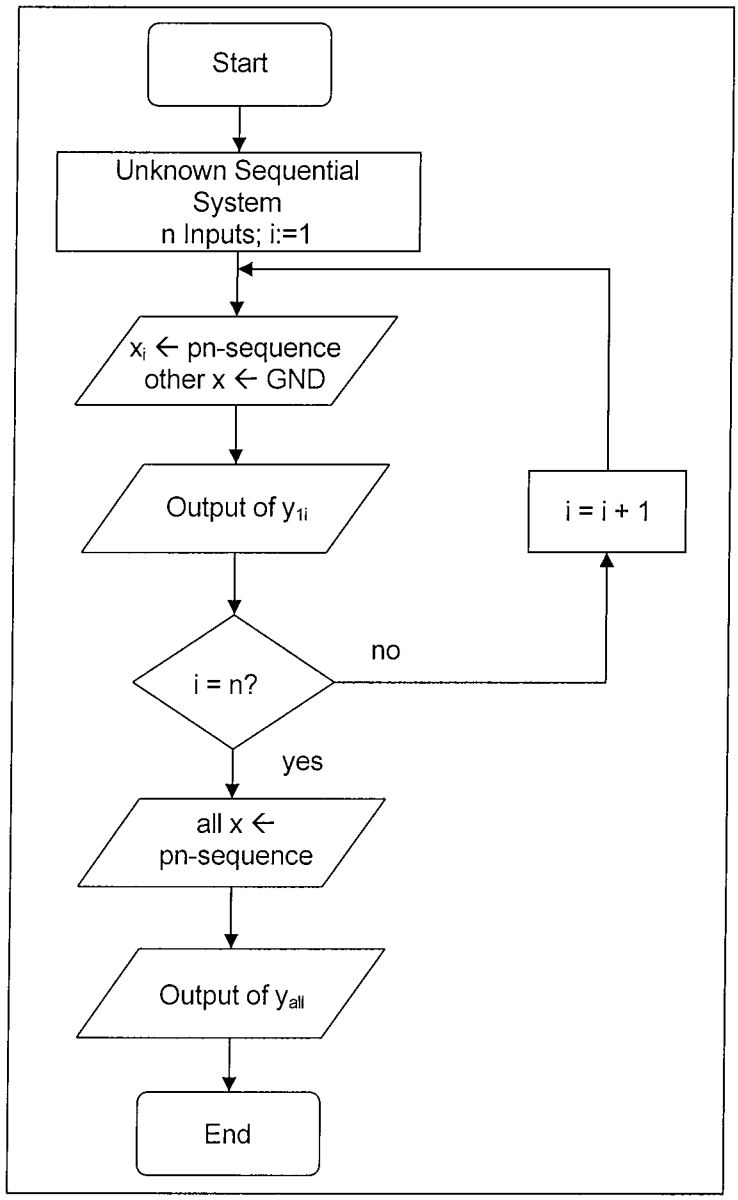

Figure 3.17: Flowchart for the Determination of Linear Behaviour

During the first analysis process the pn-sequence is applied to every input separately while the remaining inputs are grounded. During this process the impulse responses are recorded and analysed. In this example the pn-sequence is applied to the input x_1 while the inputs x_2 and x_3 are grounded. This will always be done in the same manner. In fact, a pn-sequence is applied and at the same time the remaining inputs are grounded. During this process, the impulse responses of the outputs y_1 and y_2 are recorded. This procedure is repeated for all possible inputs and the output values are investigated. Each separate output sequence is stored in a matrix. Afterwards, these result sequences in the rows are added in binary form. This means that for every

output simulation results are obtained. On the basis of this test the first results had been obtained and can be saved. These results are needed later for the classification of the unknown IC. Now, the second test cycles are carried out in a different manner. All pn-sequences are applied at the same time to all inputs. This means for the example given above, that the pn-sequence is applied to x_1, x_2 and x_3 during the same process cycles. The unknown system responds differently with respect to the first test and the impulse response of each separate output is again recorded. Next, the identification of the unknown system is carried out using these simulation results to determine whether the unknown finite state machine is linear or nonlinear. Therefore, the summed output values of the first procedure are compared with the output values of the second procedure as shown in Table 3.8.

y_1 with pn on x_1; x_2 and x_3 grounded	**y_1 with pn on x_2; x_1 and x_3 grounded**	**y_1 with pn on x_3; x_1 and x_2 grounded**	**Binary Sum**	**y_1 with pn on x_1, x_2 and x_3**
0	1	0	1	1
1	0	0	1	1
1	1	0	0	0
1	0	1	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	0
1	0	0	1	1
0	0	1	1	1
1	1	1	1	1
1	0	1	0	0
1	1	0	0	0
0	1	0	1	1
0	1	1	0	0

Table 3.8: Simulation Results of a Sequential Linear Automaton

In Table 3.8 the simulation results of the output y_1 of the example described in Figure 3.16 is given. The first three columns represent the impulse responses of the first test procedure. As can be seen, three inputs need three several test cycles. The output y_1 values are added and listed in the first four columns. The impulse response of the second test procedure is recorded in the last column. As can be seen, the results of both analysis procedures are identical. If the simulation results are identical the investigated unknown system shows a linear behaviour. Otherwise, the

unknown structure is nonlinear. In this case the unknown finite state machine must be further investigated to analyse the unknown structure of the system. It is possible, that an unknown system consists of many finite state machines which are not fully connected. Therefore, different investigations must be carried out, before the exact mathematical behaviour can be determined. The simulation results of the example have shown that the mathematical behaviour is linear. Therefore, the linear system will be investigated further in relation to the exact description of the mathematical behaviour and its particular function. The simulation results of multiple unknown linear finite states machines have shown that the identification of linear systems works accurately. After the classification of the unknown system the internal structure will be investigated to obtain exact information of the mathematical behaviour and the internal construction. Therefore, two different analysis processes are considered. The analysis of linear and nonlinear systems will be described in the following section. Since, the analysis of nonlinear systems is particularly difficult the results of linear systems are used for the investigation of nonlinear finite state machines. Therefore, in the next section linear automata will be investigated and analysed. The results of this test may be used during the analysis of other system characteristics. If the unknown system shows linear properties, then the examined circuit will be analysed on the basis of the linear analytic process as will be discussed in the next section. Otherwise, the investigated systems will be analysed using the nonlinear analysis described in detail in Chapter 4.

3.3.5 Analysis of Linear Automata

Different analysis approaches exist to investigate unknown linear systems. In this section, the mathematical background is provided to solve an unknown linear FSM. First, the theoretical cognitions are applied to an example where the results of the analysis will be discussed. Finite state machines are characterised by internal states, input sequences and the following output sequences. For this purpose, the internal states and the output function will be investigated to describe the exact behaviour of the system.

The system theory of sequential linear systems is used to analyse unknown linear FSMs [Göss72a]. Here, different aspects are considered to investigate the unknown linear system. The exact theoretical behaviour is described on the basis of the

transition and result functions as illustrated in (2.10). Again, linear finite state machines are characterised by the matrices A, B, C and D where the inputs, outputs and states are defined as vectors of a linear field. The number of result functions is given by the number of outputs while the number of transition functions is given by the number of internal states. Therefore, the determination of the transfer function $h(t)$ is a special property to describe a one-dimensional system mathematically. Based on the fact that a sequential linear system is described by means of Equations (2.10) the formulas can be simplified to determine the transfer function. If the initial state $z(0)$ of the unknown system is investigated separately, then the fundamental transition and result functions can be represented differently. This means that the transition and the result function of (2.10) will be obtained in different form as following [Göss72b].

$$z(t) = A^t z(0) + \sum_{i=0}^{t-1} A^{t-1-i} Bx(i) \tag{3.26}$$

The converted transition function is described in (3.26). The state $z(t)$ and thus, the transition function of the linear automaton is given by the initial state $z(0)$ and the input sequence $x(1), \ldots, x(t\text{-}1)$. The internal states are variable and thus, the exact determination of the transition function is not possible. The number of internal states depends on the number of storages. If the finite state machine is defined by $GF(2)$ and has n storages, then 2^n different internal states are possible. The automaton can have different states. Therefore, it is particularly complicated to determine the number of storages and consequently the current state in which the automaton is.

$$y(t) = CA^t z(0) + \sum_{i=0}^{t-1} A^{t-1-i} Bx(i) + Dx(t) \tag{3.27}$$

In (3.27) the result function is represented and it is characterised as general response equation [Wuns93]. This equation describes the output behaviour of a linear automaton in relation to the combinations of the input values. Thus, it is possible to describe the mode of operation of the unknown system by means of this input-output behaviour. The formula can be simplified on the basis of several conditions. If the initial state of the linear automaton is predetermined by $z(0) = 0$, then the equations can be reduced. Afterwards these equations may be described as illustrated in (3.28)

and (3.29). The approach simplifies the determination of unknown linear finite state machines because an initial state exists which can be used as an initial value.

$$z(t) = \sum_{i=0}^{t-1} A^{t-1-i} B x(i) \tag{3.28}$$

The expression on the right hand side of Equation (3.29) is characterised as convolution sum. This equation is particularly important and it will be used for the determination of the unknown linear structure and consequently for the characterisation of the transfer function.

$$y(t) = \sum_{i=0}^{t} H(t-i)\, x(i) \tag{3.29}$$

where $H(t)$ is defined as

$$H(t) = \begin{cases} D & if\ t = 0 \\ CA^{t-1}B & if\ t > 0, \quad H = \left[h_{ij} \right] \end{cases} \tag{3.30}$$

The behaviour of linear automata can be determined experimentally observing the input-output behaviour. In this process linear finite state machines will be considered with the initial state $z(0) = 0$. Hence, the input sequence leads to the output sequence with the help of transition functions. The linear automaton is stimulated by the impulse sequence $x(0) = 1, x(t) = 0$, where $t > 0$. This impulse sequence conforms to the Dirac function. The Dirac function is a unit impulse with the property that by means of this function the impulse response of an unknown system can be determined as represented in (3.31) [Ligh58] and [Zema65].

$$y(t) = \sum_{i=0}^{t} H(t-i)\, x(i) = H(t) \tag{3.31}$$

The finite state machine is stimulated by an impulse with the value '1' and afterwards the automaton is left to itself. The output sequence reacts to the input sequence $y(t) = H(t)$. The reaction $H(t)$ of linear automata in relation to the input sequence is denominated as impulse response or field transfer matrix for several inputs and outputs. The internal states of deterministic automata are final and therefore, $H(t)$ is periodic. The period length of $H(t)$ is given by the maximum

number of possible states. This means that the period length of $H(t)$ can be smaller or equal to 2^n if the automaton has n internal storages and is defined with a $GF(2)$. The input-output behaviour of the linear FSM is unequivocally determined by the impulse response. If the impulse response is known then the calculation of the output sequences can be executed for any sequences of input values. The derivation of the impulse response was mathematically discussed above. Now, the determination of the transition functions will be represented by means of the example in Figure 3.16. Furthermore, the description of the mathematical behaviour of linear finite state machines will be shown. The system investigated is implemented into MATLAB. In this context, the theoretical basic knowledge is used for the identification of unknown systems. In this example a linear finite state machine with four storages is used. This means that $2^4 = 16$ different states exist which the automaton can accept because it is defined with $GF(2)$. Therefore, the theoretical period length can be a maximum of 16 or less. The period length of the considered example is 7. This means, that the number of internal states can also be $2^3 = 8$.

$$h_{11}(t) = \underbrace{1011100}_{periodic} \tag{3.32}$$

As can be seen, the period length can vary and therefore, it is not possible to determine the exact number of internal storages. The knowledge about the number of internal storages is important because the number of storages is equivalent to the number of transition functions. By reason of the transition and the result functions it is possible to describe an unknown system mathematically. Therefore, the first identification of the system will be realised with the help of the impulse response. Afterwards, several further analytic processes can be operated by means of the first identification of the unknown linear system. If any input sequence is applied the impulse response describes the mathematical behaviour of an unknown system. If the linear automaton has many inputs and outputs, then different impulse responses have to be applied. This means that the matrix element h_{ij} is determined by the j^{th} input and the i^{th} output sequence. The determination of the different impulse responses is executed as follows: The finite state machine has the initial state $z(0) = 0$. The sequence $x_j = (1, 0, \ldots, 0) = 1$ is applied to the j^{th} input whereas all

other inputs are grounded using the word $x_v = (0, 0, \ldots, 0) = 0$ and $(v \neq j)$. With these input sequences the i^{th} output will result in

$$y_i(t) = h_{ij}(t) \cdot 1 = h_{ij}(t) \tag{3.33}$$

The impulse responses are measured separately for all outputs. The determination of the impulse responses is executed in the time domain. The general impulse response is identified for the output as can be seen in (3.31). Afterwards, the various output values are recorded in a matrix [Wuns93]. The matrix of several impulse responses is characterised as field transfer matrix, as illustrated in (3.34). This matrix demonstrates the coherence between input and output sequences in time domain and it describes the unknown finite state machine.

$$H(t) = \begin{pmatrix} h_{11}(t) & \cdots & h_{1r}(t) \\ \vdots & & \vdots \\ h_{q1}(t) & \cdots & h_{qr}(t) \end{pmatrix} \tag{3.34}$$

The previously considered investigations of systems are described on the basis of the example given above. The following analytic process will be carried out to determine the impulse responses. At the input x_1 the word (1, 0, ..., 0) is applied and the other inputs x_2 and x_3 are grounded with the word (0, 0, ..., 0). This means that the automaton is stimulated with a unit impulse which is labelled as a Dirac function. The system investigated responds to the Dirac function at the input with a certain output sequence. This sequence is periodically from a certain point in time. The periodical output values are the impulse sequence and are used for the identification of the system. During this procedure the impulse responses $h(t)$ of the outputs y_1 and y_2 are measured and are written into the matrix. This procedure is repeated until all possible input-output combinations and consequently all elements of the matrix are determined.

$$H(t) = \begin{pmatrix} h_{11} = 0010111 & h_{12} = 1100101 & h_{13} = 0101110 \\ h_{21} = 1001011 & h_{22} = 1100101 & h_{23} = 0010111 \end{pmatrix} \tag{3.35}$$

With the help of these impulse responses statements about the behaviour of the linear automaton can be made. For this purpose several combinations of input values are

applied to determine the corresponding combination of output values. To determine the sequence of output combinations in relation to an input sequence a convolution of input values with the transfer function must be executed. This procedure can be simplified by the application of the *D*-transfer function [Wuns86]. The *D*-transfer function must be applied to the sequence of input values and the impulse response which has also the form of a sequence. The *D*-transform of the impulse response is described as transfer function. The *D*-transferred function of a sequence $(f(t)) = (f(0), f(1), f(2), \ldots)$ is defined as

$$\begin{aligned} f^*(d) &= D\left(f(t)\right) = \sum_{j\geq 0} f(j)d^j \\ D^{-1}\left(f^*(d)\right) &= \left(f(t)\right) \end{aligned} \tag{3.36}$$

Here, D^{-1} is described as inverse transfer function. Every finite sequence which elements will be '0' from the time t_1 can be converted into a form of a finite polynomial of the order t_1-1. For an infinite sequence a formal power series can be assigned. For a periodic sequence the following holds:

$$D\left(a_0, a_1, \ldots, a_{\tau-1}, \underbrace{a_\tau, \ldots, a_{\tau+T-1}}_{periodic}\right) = \sum_{j=0}^{T-1} a_j d^j + \frac{d}{1-d^\tau}\sum_{j=0}^{T-1} a_{\tau+k} d^k \tag{3.37}$$

Introducing a main denominator and reducing common factors in numerator and denominator (3.37) can be simplified as in (3.38), where $P(d)$ and $Q(d)$ are polynomials of finite order.

$$D\left(a_0, a_1, \ldots, a_{\tau-1}, \underbrace{a_\tau, \ldots, a_{\tau+T-1}}_{periodic}\right) = \frac{P(d)}{Q(d)} \tag{3.38}$$

The *D*-transformed of discrete convolution of two sequences is equal to the product of *D*-transformed of these sequences. Therefore, from (3.31) it follows that

$$D\left(y(t)\right) = D\left(\sum_{i\geq 0} h(t-i)x(i)\right) = h^*(d)\cdot x^*(d) \tag{3.39}$$

with the transfer function $h^*(d) = D(h(t))$ and the D-transformed of impulse function $h(t)$. From (3.39) it follows that the D-transformed of output sequence is equal to the product of transfer function multiplied by D-transformed of an input sequence. The transfer function $h^*(d)$ can be written as

$$h^*(d) = \frac{P(d)}{Q(d)} \tag{3.40}$$

Again, $P(d)$ and $Q(d)$ are polynomials of finite order. For a linear automaton with a n-dimensional state vector and one input and one output ($l = m = 1$) $h^*(d)$ can be written as

$$h^*(d) = \frac{\beta_0 + \beta_1 d + \beta_2 d^2 + \beta_n d^n}{1 + \alpha_1 d + \alpha_2 d^2 + \ldots + \alpha_n d^n} \qquad (\beta_n \neq 0) \tag{3.41}$$

with $\alpha_1, \ldots \alpha_n$ and $\beta_0, \beta_1, \ldots \beta_n \in K$. From (3.41) a linear sequential circuit can be specified as shown in Figure 3.18 [ibid].

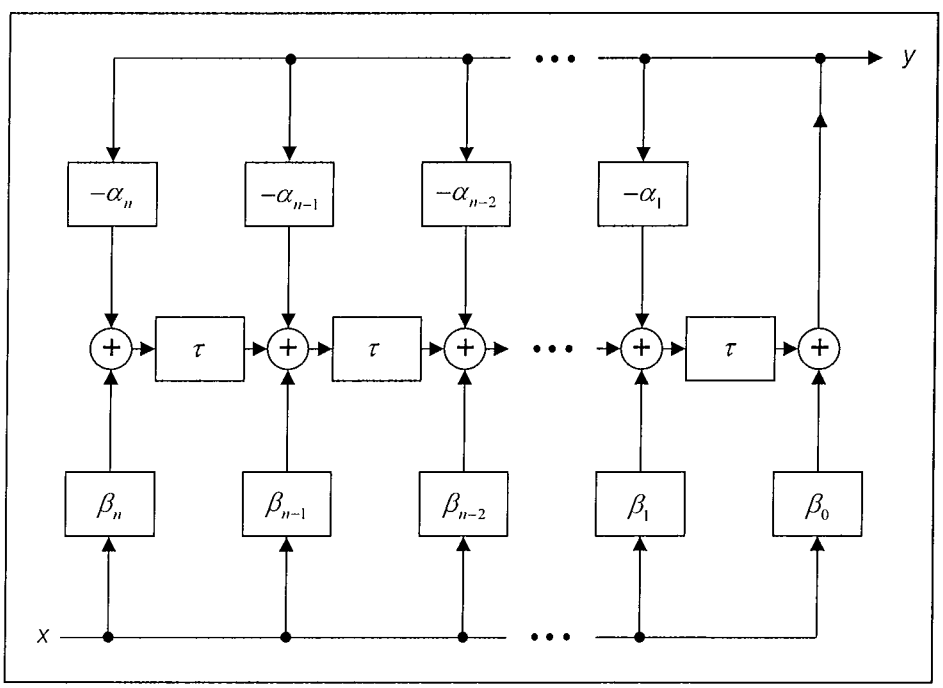

Figure 3.18: Realisation of Linear Automaton with Canonical Normal Form (3.41)

The coefficients are the multipliers of the circuit. For $GF(2)$ the minus sign can be omitted because the previous considerations can easily be transferred to linear automata with l inputs and m outputs. Here, real integrated circuits have a

multidimensional input and output structure. The response function $H(t)$ has to be substituted by the matrix $H(t) = [h_{ij}(t)]$ of the pulse responses. Instead of the transfer function $h^*(d)$ the transfer matrix $H^*(d) = [h^*_{ij}(d)]$ with $h^*_{ij}(d) = D(h_{ij}(t))$ has to be written. Equation (3.42) will be applied to the linear automaton L.

$$x(0) = \left[\underbrace{0\ldots010\ldots0}_{j}\right]^T, x(\tau) = 0 \; for \; \tau > 0 \tag{3.42}$$

Using (3.29) the outputs m will give the following answer:

$$y(t) = \left[h_{1j}(t), h_{2j}(t), \ldots, h_{mj}(t)\right]^T, \; for \; t > 0 \tag{3.43}$$

Therefore, from Equation (3.39) follows that

$$D\left(y(t)\right) = H^*(d) \cdot D\left(x(t)\right) \tag{3.44}$$

Here, x and y are vector components of the D-transformed. In Figure 3.19 a linear sequential circuit can be specified from the transfer matrix $H^*(d) = [h^*_{ij}(d)]$ [Wuns86].

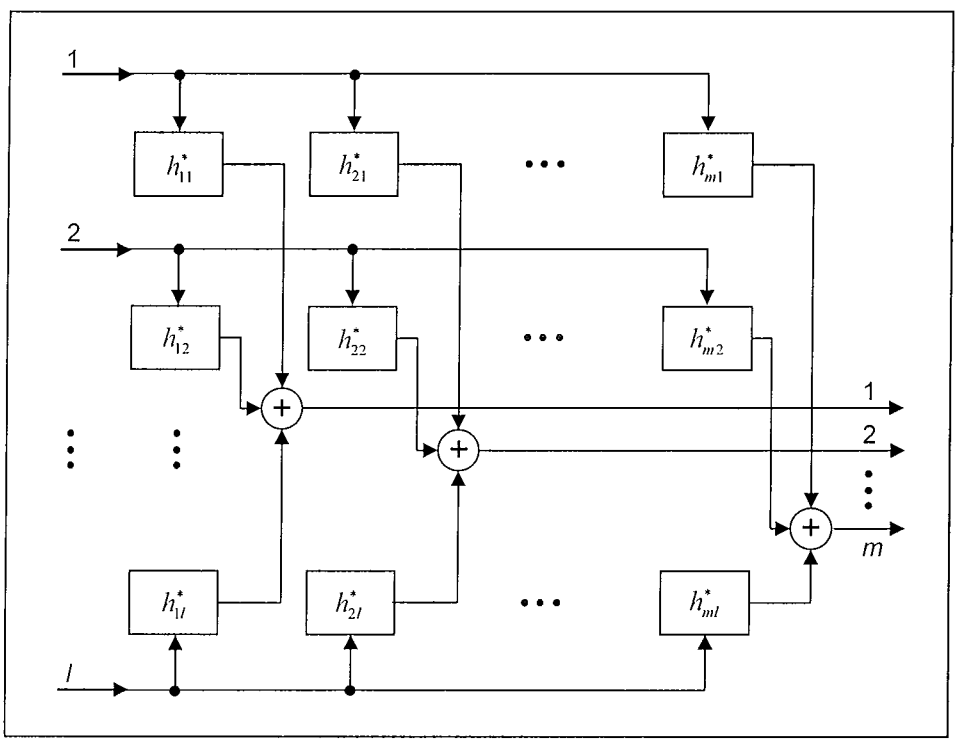

Figure 3.19: Realisation of Linear Automaton with given Transfer Matrix (3.44)

The linear automaton has l inputs and m outputs and $H^*(d)$ consists of a $m \times l$ matrix. The linear circuits with one input and one output are marked $h^*_{ij}(d)$.

Now, the identification of a sequence of output values is calculated by means of the example given in Figure 3.16 showing the analysis of an unknown system. Here, all possible internal states are unknown. Therefore, the linear finite state machine can be characterised only by the impulse response. This means that the unknown system will be described with the knowledge of the impulse response and several combinations of input values applied. The impulse response is given and on the basis of one theoretical example the determination of the output values will be represented. The theoretical results are compared to the practical analysis. Therefore, the theoretical knowledge of this section is used to describe the identification of unknown linear systems. The theoretical analytic process includes three steps. In the first step the impulse response and the input sequence are transformed to the complex frequency domain with the help of the D-transfer function. This transformation from the time domain into the complex frequency domain has the advantage that the input values and the impulse response can be multiplied. This simplifies the determination of the output sequence. In the next step, the multiplication of the transfer function and the sequence of input values will be carried out. The last step transforms the output sequence from the complex frequency domain into the time domain. Since, the input values are also generated as a finite sequence the output values will be finite and periodic. With this test the correct mode of operation of the unknown system can be verified. Since the FSM in the example consists of several impulse responses every impulse response must be considered separately. During the analysis of an impulse response in relation to one input and one output, the other input pins must be grounded. The determination of an output sequence in relation to the input sequence and impulse response will be illustrated by means of an example. For this purpose the impulse response h_{13} will be considered as represented by (3.45).

$$H^*_{13}(d) = D\left(h_{13}(t)\right) = d^2\left(\frac{1 + d^2 + d^3 + d^4}{1 + d^7}\right) \tag{3.45}$$

This example shows the D-transformed of h_{13}. Here, other impulse responses can also be described. The corresponding sequence of output values at y_1 can be calculated theoretically for any sequence of input values at x_3. For this purpose a definite sequence of input values $x(0) = 1, x(1) = 0, x(2) = 1, x(t \geq 3) = 0$ was generated at x_3 to determine the output values at y_1. The D-transform of the input values is represented by (3.46).

$$D\left(x_3\left(t\right)\right) = 1 + d^2 \tag{3.46}$$

The second step of the determination of the output sequence is the calculation of the output sequence. The sequence of input values and the transfer function are in the complex frequency domain and therefore, it is possible to multiply these functions to determine the output sequence.

$$D\left(y(t)\right) = \frac{\left(1+d^2\right)\left(d^2+d^4+d^5+d^6\right)}{1+d^7} = \frac{d^2+d^5+d^7+d^8}{1+d^7} \tag{3.47}$$

In (3.47) it can be seen that the multiplication of the transfer function and the D-transformed sequence of input values is simple. Since a is defined by $GF(2)$ it is possible that some multiplicands cancel. This property may simplify the D-transformed output sequence y. The last step is the transformation of the sequence of output values to the time domain. This transformation is executed on the basis of the polynomial division in (3.47). In this process it can be seen that a finite input sequence leads to a periodic sequence of output values because the internal number of stages are finite. The polynomial division in (3.47) was carried out and this calculation leads to (3.48).

$$y\left(t\right) = \underbrace{0100100}_{periodic} \tag{3.48}$$

The sequence of output values illustrated was calculated using a certain input sequence and the impulse response. Furthermore, the sequence of output values can be calculated for every arbitrary input sequence. This property describes the input-output behaviour.

The investigation of unknown linear automata has shown that several practical tests can be performed. Therefore, the first analysis results demonstrate that all implemented structures of linear systems can be described mathematically. Furthermore, the theoretical determination of the sequence of output values was executed to compare the findings of the practical analysis with the theoretical results. The output sequences of the theoretical and the practical test are identical. This means that the analysis of sequential linear finite state machines works correctly in relation to the input-output behaviour. The theory has confirmed the practical results. Thus, the first investigation of unknown systems has shown that it is possible to determine certain properties of unknown systems. This first information of a system can be used for the further investigation of unknown linear structures and other unknown systems. The examinations must be carried out on the basis of theoretical and practical aspects. The main feature is the determination of the number of internal states. For this purpose several combinations of input values will be applied to determine the mode of operation. Unknown linear systems consist of several internal structures. Therefore, the investigation cannot be defined by the investigated structure, but other possible linear structures must also be examined. Since linear systems are a special case of the fundamental sequential finite state machines it is particularly important to apply these findings to other unknown systems. The analysis of linear FSMs was discussed in this section. In Section 3.3.6 the determination of state numbers will be introduced for non-recursive linear sequential FSMs which are a subgroup of sequential FSMs.

3.3.6 Determination of State Numbers

In this section a modification of the cross-correlation function (CCF) is given which can be used to determine the number of states in non-recursive sequential unknown ICs. Traditionally, the CCF will be used to determine time-delays in signal and system theory or fibre optic measurement systems [Rich99] [Bund05]. For example the principle of backscatter analysis and backscatter fibre optics atmospheric sounding is based on this correlation method [ibid]. This idea is taken and transferred for the determination of linear non-recursive FSMs. In Figure 3.20 the procedure is illustrated in principle to determine the number of internal states $z(t)$ where $x(t)$ describes the input values and $y(t)$ describes the output values.

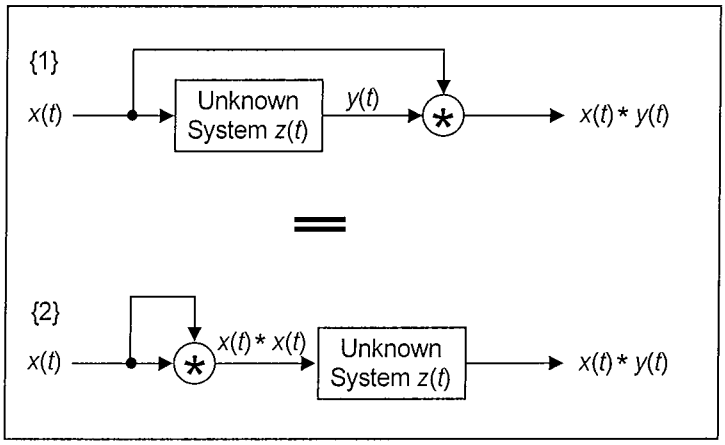

Figure 3.20: Determination of State Numbers

In Figure 3.20 it can be seen that the upper schematic {1} is based on the investigation of the CCF of input signal $x(t)$ and output signal $y(t)$. This schematic is equivalent to a stimulation of the unknown system with a signal corresponding to the auto-correlation function on the bottom schematic {2}. Therefore, the ACF $x(t) * x(t)$ can be realised using a pn-sequence generator as was described in Section 3.3.1. Again, this pn-sequence is realised using a linear feedback shift register. The following conditions have to be considered to use the measurement system above. The period of the generated pseudo noise sequence must be greater than the number of possible states in the unknown system to avoid repetitions of the pseudo noise sequence. Otherwise, this could lead to false results during the computation of the cross-correlation function.

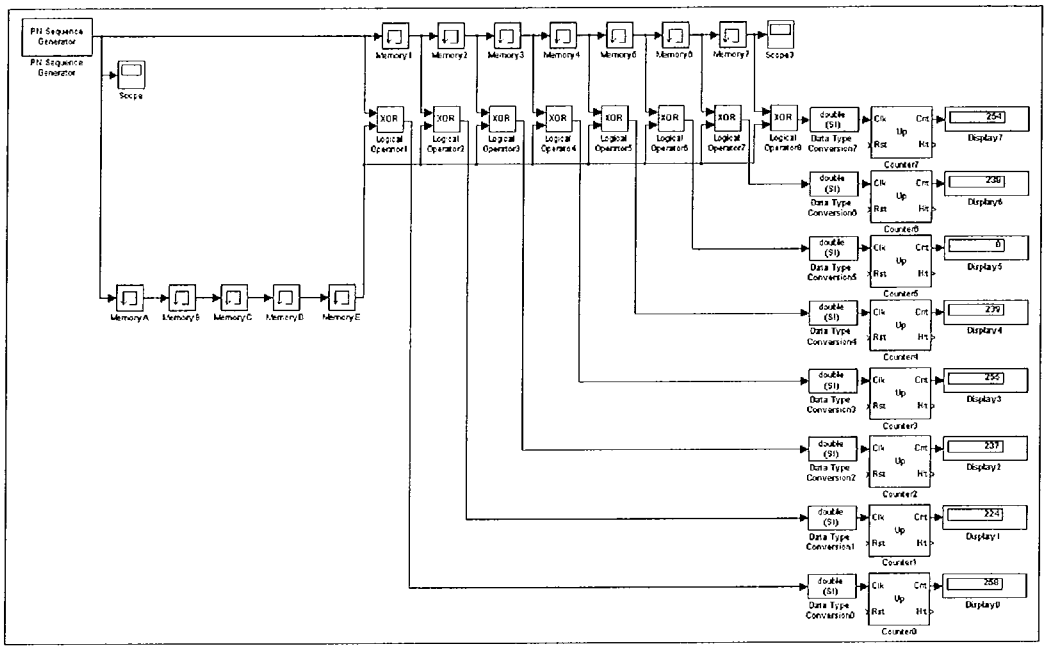

Figure 3.21: Example of an Implemented Linear FSM

As illustrated in Figure 3.21 the cross-correlation function is realised using shift registers consisting of D-latches. The number of D-latches must be greater or equal to the number of internal states in the unknown system to evaluate the number of states correctly. This is necessary because the CCF calculates only the correct values if the signal propagation delays through the internal states and the D-latches are identical. The CCF is a summation which is realised using up/down-counters. The outputs of the D-latches are separately linked modulo-2 with $y(t)$. In the example described above the minimum result of all counters leads to the number of D-latches. Here, the 5^{th} counter displays the minimum. Consequently, this number is five and is equal to the number of states in the unknown sequential finite state machine.

3.4 Summary

This chapter has presented the analysis process to investigate an unknown IC. In Section 3.1 a general overview of the novel analysis procedure developed was given. Furthermore, it was shown that it is suitable to divide the overall analysis into three main parts, the determination of pin types, the preliminary investigation and the identification of the unknown IC. Section 3.2 then discussed the determination of pin types which was explained in detail using the IC 74 HCT 00 as an example. Here, the electrostatic discharge phenomenon was used to identify the input, output,

voltage supply as well as ground pins. The knowledge of pin types and their location is important for the further analysis. After having described the determination of pin types the separation procedure was introduced in Section 3.3 which also consists of three main parts. Here, the determination of independent blocks, the division into combinatorial or sequential FSMs and the division into linear sequential or nonlinear sequential finite state machines were explained in detail. However, to avoid a false interpretation of the IC to be investigated the determination of independent blocks was explained in Section 3.3.1. Here, the use of binary maximum sequences is the key to separate the independent circuits inside the unknown IC. The same sequences can be used to classify the IC into combinatorial or sequential FSMs which was described in Section 3.3.2. Thus, if the investigation has resulted in combinatorial behaviour in Section 3.3.3 the identification of combinatorial FSM was discussed. A novel procedure based on the superposition principle was described in Section 3.3.4 which can be used to classify sequential FSMs into linear or nonlinear behaviour. Again, the same binary maximum sequences previously derived can be used. If the analysis has resulted in linear behaviour Section 3.3.5 then presented the identification of such unknown ICs. If the IC under investigation has non-recursive linear behaviour the cross-correlation function of Section 3.3.6 can be used to determine the number of states. However, a large number of unknown ICs have nonlinear behaviour. Therefore, the following chapter will discuss the investigation of such finite state machines.

4 Analysis of Nonlinear Finite State Machines

The previous chapter presented the analysis of different types of unknown integrated circuits. It was shown that the novel separation procedure introduced in Chapter 3 has resulted in a correct identification of the unknown ICs under investigation according to its behaviour. Then, the analysis procedures for combinatorial and sequential FSMs were discussed in detail. It was further shown that if the classification procedure detects nonlinear behaviour the analysis has to be further continued. Therefore, new strategies to non-destructively identify nonlinear deterministic finite automata will be explained in this chapter. Firstly, a general overview of related work will be given. Then, in Section 4.2, the developed algorithm will be explained in principle. After that, the separation into Moore or Mealy automata will be described. The overall solution procedure can be divided into a preparation algorithm and the main algorithm which are presented in Section 4.4 and Section 4.5 respectively.

4.1 General Overview

The majority of today's ICs are nonlinear systems. Consequently, the main focus has been on solving such FSMs. As described in Section 1.2.2 Moore first proposed the identification of automata. To illustrate this, Moore's algorithm is now explained [Moor56]. For this, test sequences have to be generated to identify a given machine with no more than r states. Therefore, all machines with r states, m inputs and l outputs have to be constructed by considering all possible next states and outputs of each mr transitions. A machine with less than r states is equivalent to a machine with r states and therefore, it has not to be considered. The total number of mr transitions, $\boldsymbol{R}$, can be easily checked with Equation (4.1).

$$\boldsymbol{R} = \frac{(rl)^{mr}}{r!} \tag{4.1}$$

Every machine which is not strongly connected has to be discarded. For the remaining finite state machines only one has to be kept in a class of equivalent

machines obtaining a set of reduced, not equivalent and strongly connected machines with no more than r states (4.2).

$$M_i = (\underline{X}, \underline{Z}_i, \underline{Y}, g_i, f_i) \qquad i = 1, ..., R \text{ where } R < \boldsymbol{R} \tag{4.2}$$

The implemented finite state machine A is equivalent to one and only one of the machines and has to be identified. Then, a direct sum machine M of the R component machines has to be constructed as follows. Machine M has the same input set $\underline{X}$ and output set $\underline{Y}$ and its states are the union of the states of all the component machines as in (4.3).

$$\underline{Z} = \bigcup_{i=1}^{R} \underline{Z}_i \tag{4.3}$$

The transition function and the output function g and f are natural extensions of the respective functions of the component machines for an input $a \in \underline{X}$ and state $z \in Z_i \subset Z, g(z,a) = g_i(z,a)$ and $f(z,a) = f_i(z,a)$. Obviously, machine M is reduced and A is equivalent to a component machine M_k which has to be identified. It is supposed that A is in an initial state that is equivalent to state z_k in M_k, which is also unknown. However, if it is possible to apply a test sequence to M and determine the final state then the containing component machine must be M_k since any input sequence takes the machine from z_k to a state also in the same component machine M_k. A homing sequence h of M would satisfy this purpose. The sequence h has to be applied to the implemented machine A. Next, the final state z_l in M will be determined from the outputs from A. Now, the containing component machine $z_l \in M_k$ can be identified, which is equivalent to A. The implemented FSM A is now identified from its input-output behaviour. A homing sequence has in general quadratic length in number of states. However, in the direct sum machine there is one length $\underline{Y}(r^2R)$, which is much smaller than R^2 because R is exponential in r. The reason is that any of two states of M can be separated by a sequence of a length of at most $2r$-1. This separating sequence can be obtained by applying the minimisation procedure to the direct sum of at most only two component machines that contain the two states. Again, the homing sequence is constructed by combining no more than rR which is the number of states of the machine M like pair wise separating sequences and has the length $\underline{Y}(r^2R)$. Constructing the homing sequence h and then applying it to the direct product machine M the complexity is quadratic in R because it

involves tracing the behaviour of M starting from all the states on input sequence h. Moore's algorithm is only valid for the Moore automaton. In general, Moore uses the verification with a known FSM.

Another interesting paper was presented by Hennie [Henn64]. If the FSM has a distinguishing sequence of length L then it is possible to construct a checking sequence of polynomial length in L and the size of machine. Unfortunately, not every machine has a distinguishing sequence. Furthermore, only exponential algorithms were known to determine the existence and to generate such sequences. Hennie also gave another nontrivial construction of checking sequences in the case that a machine does not have a distinguishing sequence [ibid]. However, Hennie's checking sequence is exponentially long. From his work it can be concluded that fault detection is easy accomplished if the FSM has a distinguishing sequence. Otherwise, it is a more difficult problem. In the 1960's several papers were published on testing problems motivated by automata theory and testing of switching circuits. Kohavi presented a overview of the major results [Frie71] [Koha78]. During the late 1960's and the 1970's there were a lot of activities in the Soviet literature. Vasilewskii presented an important paper on fault detection [Vasi73] where he proved polynomial upper and lower bounds on the length of checking sequences. He showed that for a specification FSM with r states, m inputs and l outputs checking sequences exist which are presented in Equation (4.4).

$$length_{CS} = \underline{Y}\left(m^2 r^4 \log(lr)\right) \tag{4.4}$$

Furthermore, there is a specification finite state machine which requires checking sequences of length as in (4.5).

$$length_{CS} = \Omega\left(mr^3\right) \tag{4.5}$$

However, the upper bound was obtained by an existence proof and he did not present an algorithm to generate efficient checking sequences. Chow then developed a method to construct a checking sequence of length as in (4.6) [Chow78].

$$length_{CS} = \underline{Y}\left(pr^3\right) \tag{4.6}$$

After describing checking sequences the genetic algorithm will be considered. It is another possibility to abstract unknown ICs. However, it is based on the theory of evolution and the principle of the survival of the fittest [Heis94]. The main advantage of the genetic algorithm is its use for a variety of different combinations and variable sizes of the circuit. Its disadvantage is mainly caused by the fitness function and therefore, by the necessary mutations and heredities. The function would have to be implemented in a way that allows identifying the individuals who are adapted best without major misinterpretations. However, the whole examination of single states is not feasible because the states cannot be addressed directly. It is only possible to check single paths to compare them with the parallel running automaton. Furthermore, the number of corresponding paths would need to be weighted to decide about their conformance. This step by step operation would cause a large amount of information to be processed. In addition, it would be important to minimise the modifications in the conforming information by mutation and heredity or eliminate them entirely. Additionally, it would be necessary to store this data.

Other verification procedures can also be exemplary found in [Schl02] [Musl03]. In [Schl02] a procedure to verify a black box system is described. Here, it is checked if an unknown system contains predefined properties of an already known system. The entire identification of the unknown system is not the aim of the procedure described in [ibid]. A procedure of formal verification of automata is presented in [Musl03]. Here, the unknown automaton is compared with a known automaton with respect to its occurring states. However, using this procedure an overall IC analysis is not possible. The aim of this research was to develop a procedure to identify nonlinear unknown FSMs without any prior automaton knowledge. The next section discusses the preliminary considerations.

4.2 Preliminary Discussion

Normally, unknown digital ICs to be investigated consist of many logic circuits as well as internal storages. Moreover, these storages can be composed of individual flip-flops. Here, each bit combination of the internal storage relates to one state of the automata. The state transition is carried out by applying a bit combination as at the inputs of the automaton. Next, a clock is then applied to process the input word.

Now, the resulting next state depends on the current state as well as the input word applied during clock pulse. However, if in each case the initial state as well as the applied input word are the same then the same state can always be reached. If the complete sets of input words, state transitions as well as output words are recorded an unknown automaton can be fully described. In the following the overall nonlinear analysis will be presented.

The overall algorithm which has been developed for the solution of analysis of nonlinear FSMs consists of three parts. These three main steps are the separation into Moore or Mealy automata, the preparation of the main algorithm and the main algorithm itself. Both the preparation of the algorithm and the algorithm itself consist of two similar parts which represent the particular solution for Moore and Mealy automata. Moreover, the algorithm was designed in a way that allows using it for an fully variable amount of inputs, outputs and states. The used simplification is the knowledge of the pin assignment and in particular the use of clocked resettable automata. Additionally, it has to be mentioned that the solution found was designed for the synchronous subset of the nonlinear circuits which is a realistic assumption for the investigation of real integrated circuits. The general analysis of nonlinear finite state machines is illustrated in Figure 4.1.

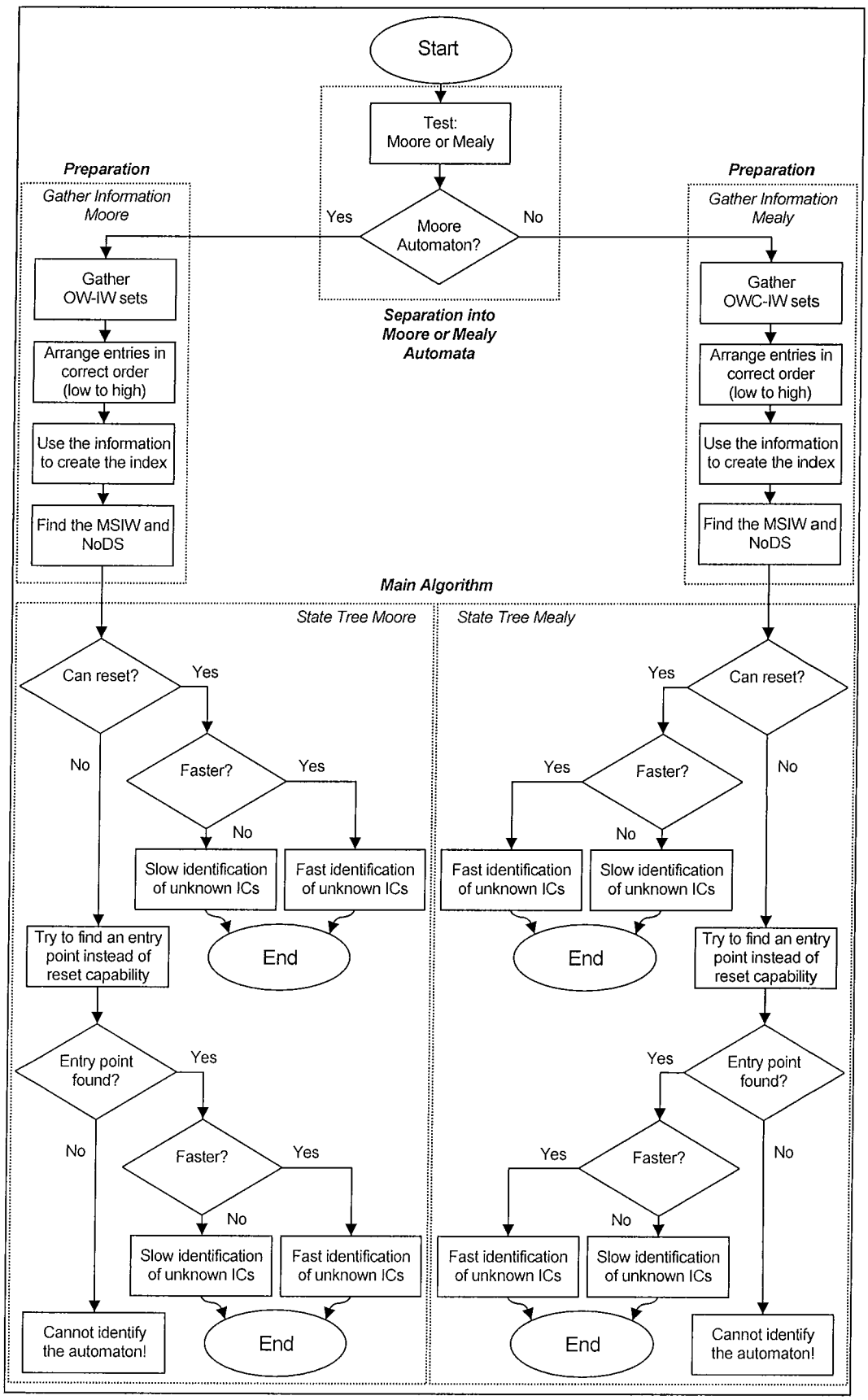

Figure 4.1: The Analysis of Nonlinear FSMs in Principle

As can be seen in Figure 4.1 the separation into Moore or Mealy automata is the first part of the overall algorithm which will be explained in the next section.

4.3 Separation into Moore or Mealy Automata

The effective separation into Moore or Mealy is an important improvement of the procedure to other methods described. Here, it is possible to discriminate the considered nonlinear FSMs into two groups. As was previously described in Chapter 2, automata can be divided into Moore or Mealy automata. As already shown the output of a Moore automaton only depends on the internal states as illustrated in Figure 2.16 [Moor56]. Additionally, if the storages and the inputs are connected by combinatorial circuitry the automaton is of type Mealy as shown in Figure 2.15 [Meal55]. Most of today's integrated circuits are a mixture of both, by having some outputs that work as Moore and others that work as Mealy. Such automata are described as pure Mealy automata. The basic classification into Moore and Mealy is reasonable and necessary, since the analysis of a Mealy automaton is more complex than for a Moore automaton. Furthermore, it was found that the complexity of a Mealy automaton is actually very beneficial for its analysis. While a Moore automaton has only one output word per state, a Mealy automaton as illustrated in Figure 2.18 has a whole set of output words [ibid]. Therefore, the number of output words per state is similar to the number of input words. Each of these output words is gathered by applying an input word to the circuit without causing a clock. This variety of output words provides far more information, which can be used to identify the states clearly. This is comparable to the names of persons. If somebody has only one name, which in addition has a fixed number of letters according to the number of outputs used, it is likely that somebody else is called the same. Having many of these names would lower this probability dramatically. To test automata for being Moore or Mealy all input words are applied to random states without causing a clock. If the output word changes it is a Mealy automaton. The examination of every single output associated to Moore or Mealy category and analysing these parts separately was investigated. However, it was seen that this procedure would not make sense, since significant parts of the output words would have been cut off. In general a Moore automaton is only able to benefit from the number of outputs like the letters in the name. A Mealy automaton additionally

benefits from the number of inputs which is equal when compared to the quantity of names. Since all of these outputs belong to the same set of states they should share as much information as possible. The procedure is described by the flow chart shown in Figure 4.2.

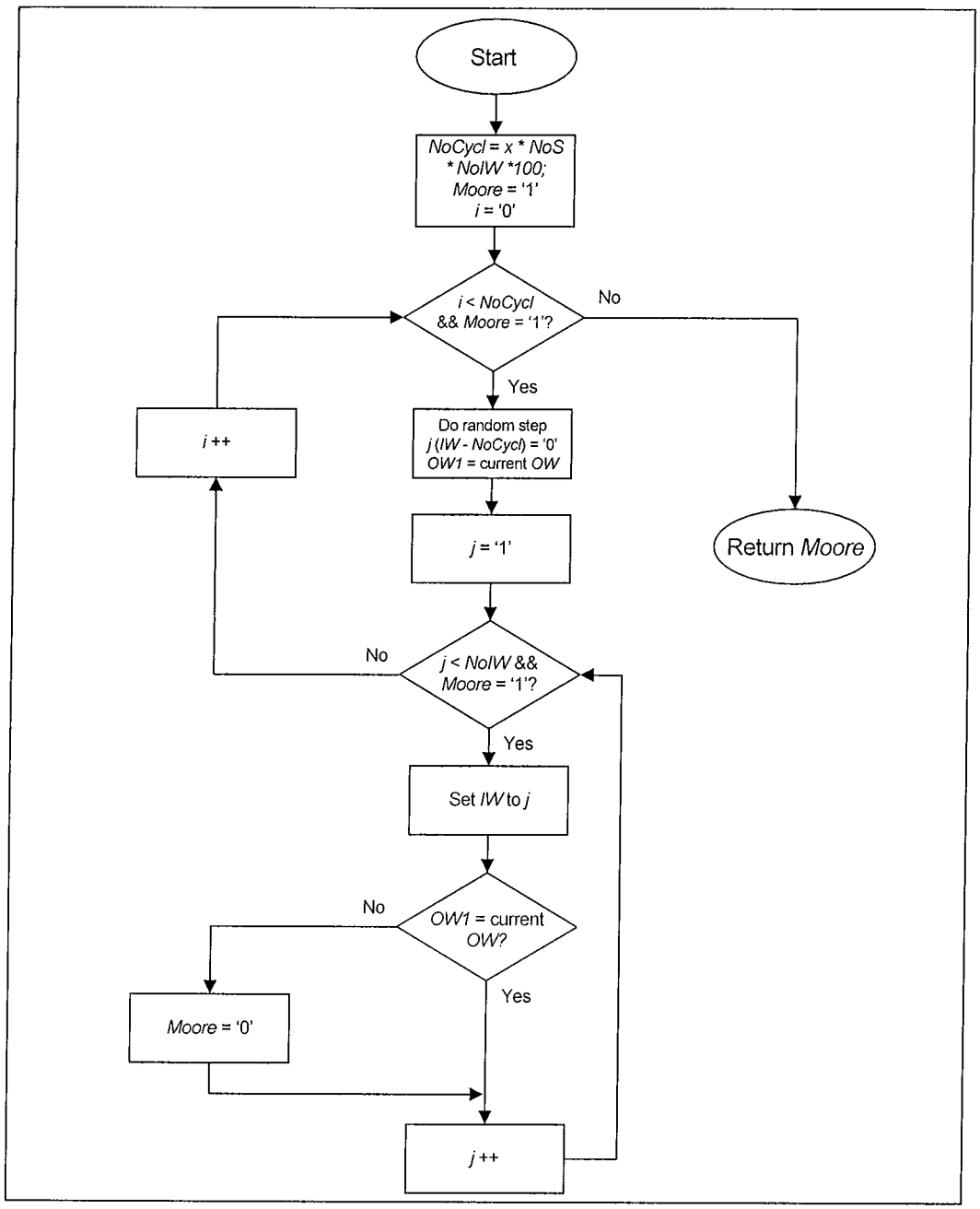

Figure 4.2: Separation into Moore or Mealy

The number of cycles (NoCycl) is a counter variable specifies how often the separation procedure has to run. Moreover, the number of cycles using random input words is used to reach as many states as possible, even if it is not likely for a Mealy

automaton to cause the same output words with every input word at many of the states. Before the number of cycles can be calculated the number of input words (NoIW) has to be determined. The number of input words can be determined as in Equation (4.7).

$$NoIW = 2^{NoUIP} \tag{4.7}$$

where the number of used input pins (NoUIP) is a result from the determination of pin types. The higher number of input words the higher the number of cycles. Therefore, it also influences the switching activity of the FSM under investigation. Depending on the actual knowledge about the internal states there are two possibilities to set the number of states (NoS). If the number of states is completely unknown the default value NoS = 0 has to be set up. Then the algorithm uses a new iterative approximation described later on to find the real number of states. If the user already knows the exact or the maximum number of states he can enter this value in the analysis environment before starting the analysis which will be described in Section 5.1. The advantage is that in this case the algorithm can be simplified and uses less computing time. If NoS = 0 which is the standard setting then the following Equation (4.8) holds.

$$NoCycl = NoIW \times 10000 \tag{4.8}$$

Here, the number of input words is multiplied by 10000 to ensure the detection of state transitions. If the number of states is already known Equation (4.9) shows the number of calculated cycles.

$$NoCycl = NoS \times NoIW \times 100 \tag{4.9}$$

Here, the number of states directly influences the number of cycles. In (4.9) the number of input words is multiplied by 100 to ensure the detection of state transitions.

Initially, the automaton is classified as a Moore automaton. The test run is now started and a random step is made. Making a step means to apply an input word and then causing a clock pulse. A random step is made if the input word is chosen by a random generator. After this step, the first input word ('0') is applied to the

automaton and the resulting output word is stored. In the following loop all other input words are applied to the circuit, but no pulse is caused. The output words which appear are compared to the stored one and in case of a variation the automaton is classified as a Mealy automaton and the procedure is finished. If all output words are identical the next random step is made until the number of cycles is reached. If the last cycle is executed and no variance was found, the automaton will be considered as a Moore automaton further on. The investigation of Mealy or Moore automata is basically the same and will be described in the following using a Mealy automaton. Whenever a output word combination using a Mealy automaton is mentioned in this thesis this means a particular output word using a Moore automaton. In the next section the preparing algorithm will explained which is the second part of the overall nonlinear analysis procedure.

4.4 Preparing Algorithm

After the separation into Moore or Mealy automata the main algorithm is prepared by process steps. This preparation is an important task for an efficient mode of operation of the analysis. However, the identification procedure must firstly find an initial state of the automaton. In the simplest case the initial state can be reached by a reset pin control. In other ICs the reset can be carried out by short disconnection of power supply which is also called power-on-reset. However, an initial state can be also found using suitable process steps if such a reset capability does not exist. If the initial state is found then the nonlinear analysis can be carried out. The preparing algorithm is now explained. Firstly, the information about the input-output words or input-output word combinations respectively is gathered. These sets are recorded as described in Equation (4.10).

$$\begin{aligned} &\{1\} \quad OW(t-1); IW; OW(t) \\ &\{2\} \quad OWC(t-1); IW; OWC(t) \end{aligned} \tag{4.10}$$

If the analysis has identified a Moore automaton set {1} is used. Set {2} is used in the case of a Mealy automaton. Figure 4.3 shows the four main steps of the preparing algorithm in principle.

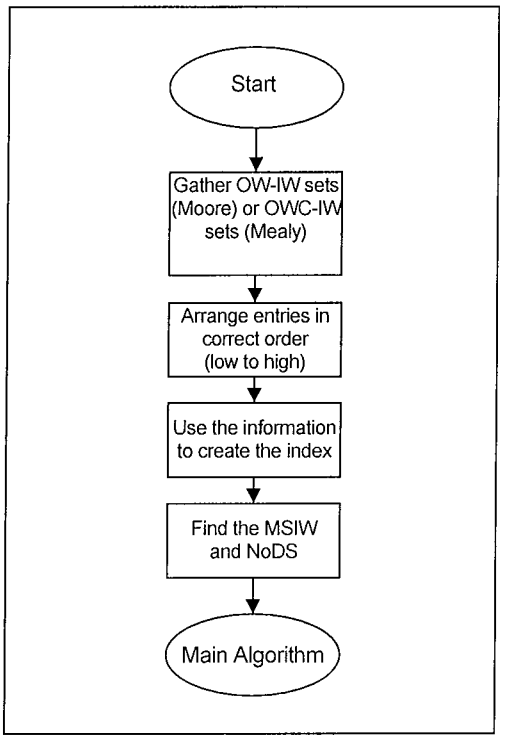

Figure 4.3: The Preparing Algorithm in Principle

After the recording of information sets the entries are sorted from low to high and an index is generated. Considering this information at the end of the preparation the most significant input word (MSIW) has to be found out. The most significant input word is recorded for later use. Next, the number of distinguishable states is calculated. In the next section the gathering of the information sets as well as the generation of the index will be explained in detail.

4.4.1 Gathering Information and Generating the Index

After the type of the automaton was determined all occurring output words for a Moore automaton or output word combinations for a Mealy automaton are gathered. Again, the following steps and parts of the algorithm are explained for a Mealy automaton and are carried out in an analogous manner for Moore automata. However, a single output word (Moore) is processed instead of all output word combinations (Mealy). The different states are separated relative to their obvious differences in their output behaviour before the algorithm is started. Therefore, random input words are applied and a clock pulse is generated afterwards. In regular intervals a reset is applied to resettable automata to restart the algorithm from well

defined initial states. The previous and the following output word are recorded at each step. Additionally, the input word which has caused the step is stored. Repeating combinations of these three values are not saved. However, all differences are collected using the described procedure. As always the first *OW(t-1)* the change of the *OW(t)* to multiple applications of different input words is investigated and is used for the result. If the current output word has two or more following output words when applying the same input word then the number of these output words relates to the minimum number of states which share the first output word. This is valid for deterministic automata. Each information set as illustrated in (4.10) is checked if it has already occurred. If it has not it is added to the current list. The number of investigation steps is the same as the number of cycles already calculated in (4.8) or (4.9) respectively. The number of found output word is stored as Found Number of Output Words (FNoOW). The number of states, the output words, the input words and the type of the automaton are the basic information. The combinations are checked for their first output word. Due to their order each alteration implies a new output word. An example of a Mealy automaton is shown in Table 4.1.

OWC (t-1)			CNo	IW	OWC (t)			CNo
0	...	NoIW – 1	0	0	0	...	NoIW – 1	1
0	...	NoIW – 1	0	0	0	...	NoIW – 1	3
0	...	NoIW – 1	0	1	0	...	NoIW – 1	3
0	...	NoIW – 1	0	2	0	...	NoIW – 1	0
0	...	NoIW – 1	0	2	0	...	NoIW – 1	2
				.				
				.				
				.				
0	...	NoIW – 1	0	NoIW – 1	0	...	NoIW – 1	0
0	...	NoIW – 1	1	0	0	...	NoIW – 1	1
				.				
				.				
				.				
0	...	NoIW – 1	FNoOW – 1	NoIW – 1	0	...	NoIW – 1	2

Table 4.1: Data Structure of the Preparing Algorithm

Here, CNo is the combination number and NoIW is the number of input words. These alterations are counted to find out the number of occurring output words. If the automaton is of type Mealy, then two of them will have one more dimension for every input word to store all combinations. All information are recorded in the array

and afterwards compared and arranged. The columns named CNo form a simplification which is explained more detailed later on. The entries in the right column CNo in Table 4.1 are examples, also the number of recurrences of the input words. Mainly due to the amount of redundant information that is created in Mealy automata by the quantity of identical input words, the combination number assigns an alias to each output word combination. This allows the reduction of the number of stored digits to one, but an index is required to manage the synonyms. This index has been extended by some more information. It is illustrated in Table 4.2.

CNo	OWC			Significance			MSIW	Position
0	0	...	NoIW - 1	0	...	NoIW - 1	0	0
1	0	...	NoIW - 1	0	...	NoIW - 1	2	8
					.			
					.			
					.			
FNoOW - 1	0	...	NoIW - 1	0	...	NoIW - 1	1	44

Table 4.2: Data Structure Index

The entries in MSIW and position are examples. For each input word the significance shows how many different following output word combinations for the same first output word combination are caused by this input word. It is used for the determination of the number of distinguishable states and is rejected afterwards. The most significant input word stores the input word with the highest significance. The arrays containing the redundant information are discarded. These indices are used for the Moore automata, too. Even if they cannot reduce the required memory they still allow a more systematic processing and examination of the lists. Such a case implies the existence of several states with the same output word. If the combination is a new one it is appended to the list. During the whole process a counter is running which stores the number of entries. Since the output word combinations do not occur in a particular sequence they will be rearranged in ascending order to ease subsequent processing. After describing the first steps of the preparing algorithm the determination of the most significant input word will be presented in the next section. Furthermore, the calculation of the number of distinguishable states will be given.

4.4.2 Determination of the Most Significant Input Word and Calculating the Number of Distinguishable States

The input word, which causes most output words following a particular output word, is labelled as most significant input word. Hence, the minimum number of all states is determinable because each output word found is investigated in the described way. Again, there is no targeted search for an output word which means that the gathered output words are caused by the randomly applied input words. Furthermore, this number is the number of assured distinguishable states. With the help of the most significant input words it is possible to separate states that have the same output word and the same input word is applied but the following output word is a different one. For instance, if there would be three such entries there will be at least three states related to this output word. If the significance of all most significant input words is added they will reveal the number of states that can be determined securely. Equation (4.11) shows the calculation of the number of distinguishable states (NoDS).

$$NoDS = \sum_{MSIW} significance \tag{4.11}$$

If this number is equal to the number of states, then the automaton can be identified directly. Otherwise, the number of states equal to the difference between the number of all states and the number of distinguishable states has the same output word as other states and it is not possible to determine the automaton in only one step. If entries exist where at any time input words and output words are equal but the following states are different, then the list is rearranged. The detection of such entries is a clear proof that a minimum of two states exists with the same output word. The total number of finally found differences forms the number of securely distinguishable states. This means, that it is possible to compare two sets of $OW(t\text{-}1)$; IW; $OW(t)$ for a Moore automata or $OWC(t\text{-}1)$; IW; $OWC(t)$ for a Mealy automata there is no difference in the actual sets but only in the result of the investigation. The use of a number of distinguishable states severely reduces the necessary investigation depth of the integrated circuit. The classification into Moore or Mealy automata as well as the consideration of the number of distinguishable states is an important part of the analysis procedure. The use of the number of

distinguishable states is only necessary to determine the solvability of non-resettable automata. After having explained the preparing algorithm the main algorithm will be explained in the next section.

4.5 Main Algorithm

The main algorithm is the major part of the analysis procedure for nonlinear FSMs. It consists of several parts which are explained in detail in this section. The procedure works similarly for Moore and Mealy automata. Figure 4.4 schematically shows the main algorithm. If the unknown IC is not identified yet the required length of the investigation tree is first determined. If the IC under investigation has a reset capability then the algorithm queries the solution type in the next step. These are the fast or the slow identification. Basically, both the fast and the slow analysis are equivalent. However, the slow identification is a subset of the fast identification. In case of limited user memory it is not possible to process complete state trees with maximum tree length. Here, the algorithm automatically switches to the slower solution. The identification will be explained in detail in Section 4.5.1 while an example is given in Section 4.5.2. The features of the slow analysis are given in Section 4.5.3 after the complete identification of the unknown IC the algorithm is finished. If a non-resettable unknown IC has to be analysed the main algorithm tries to find an entry point instead of an initial state as will be reached by the reset capability. This will be explained in Section 4.5.4.

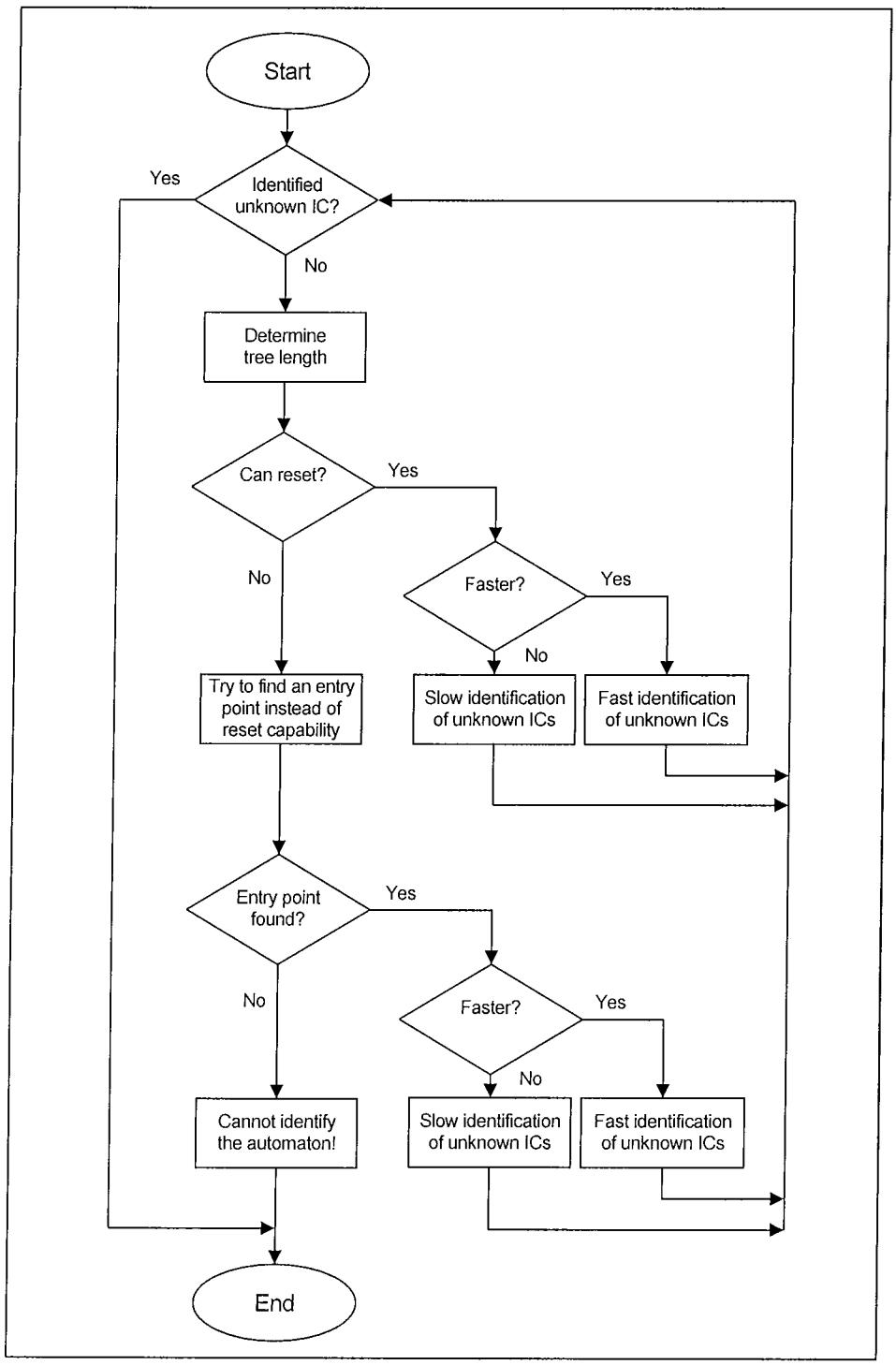

Figure 4.4: The Main Algorithm in Principle

If it is possible to find an entry point then the analysis goes to the similar identification using fast or slow analysis. Here, for each reset the initial state has to be found using a combination of random and controlled steps. If no entry point can be found the unknown automata cannot be identified. After describing the main

algorithm in principle the identification procedure will be presented in the next section.

4.5.1 Identification of Unknown ICs

The maximum number of states needs not to be known to proceed the algorithm. In most cases the number of states (NoS) can be calculated using iteration. The initial value can be either given by the user or is determined from the number of distinguishable states (NoDS). If the real number of states is not known the number of distinguishable state for the initial value can be calculated as in (4.12).

$$NoS = NoDS + 2 \tag{4.12}$$

The added two is based on the matter of fact that it is the number of guessed and not identified states. This number can be chosen in a free range. If a higher value will be chosen the likelihood increases to find distinguishable states in each identification cycle which would previously have not been detected. At the same time the investigation complexity increases. If the addend '2' of Equation (4.12) is too high this advantage could be lost and converted into a disadvantage. The iteration to the real number of states is carried out after each cycle of the main algorithm. In the simplest case the number of distinguishable states as in Equation (4.12) is replaced by the number of found states (NoFS) as in Equation (4.22).

$$NoS = NoFS + 2 \tag{4.13}$$

Again, the addend two is a selectable value which function was previously described. To illustrate the operation of the analysis Figure 4.5 shows the slow and fast identification of unknown IC more detailed. Here, it has to be mentioned that the fast and slow identification are related to each other.

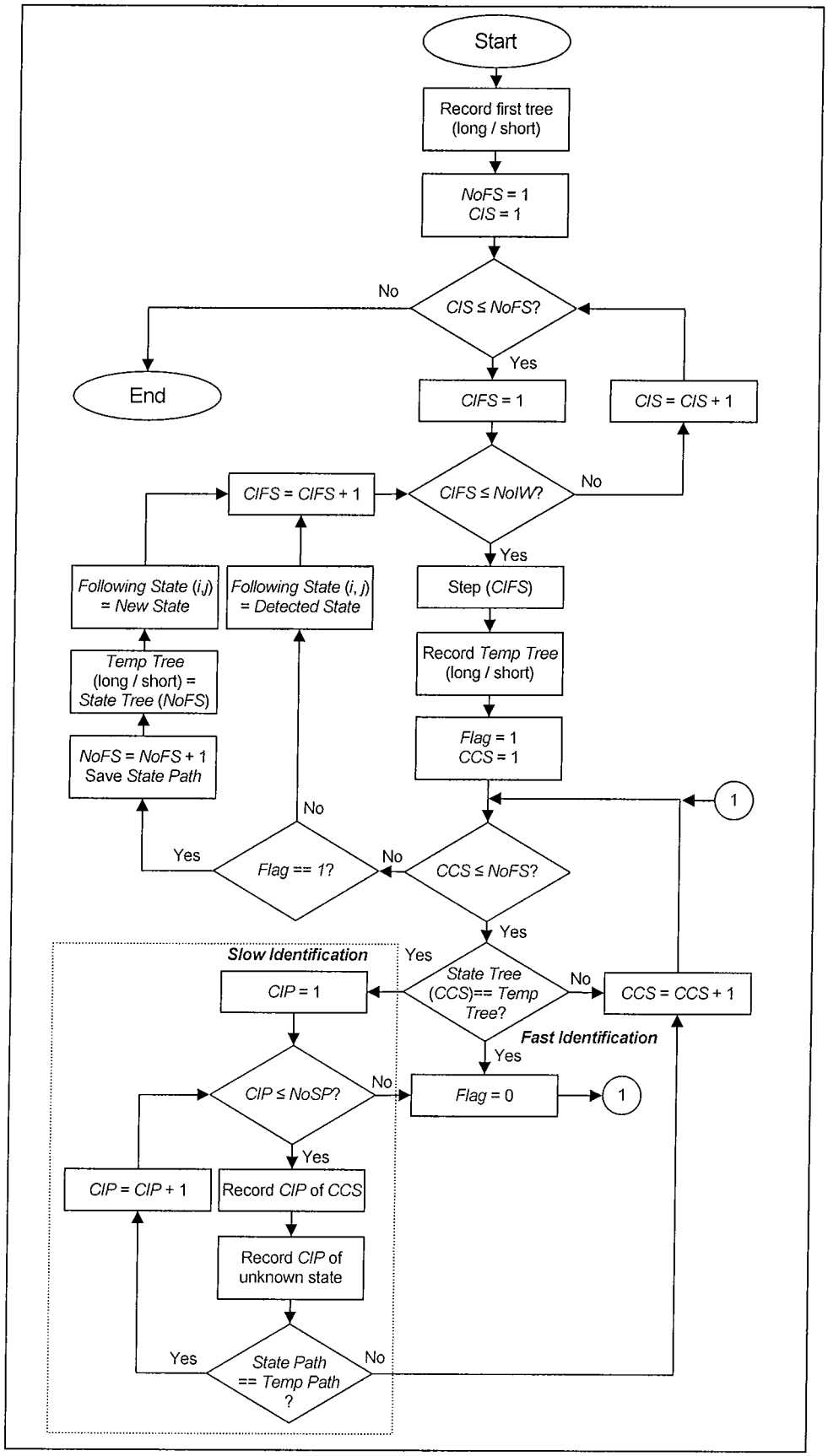

Figure 4.5: Identification of Nonlinear FSMs

The idea behind the fast identification of unknown ICs is similar to the general classification of states using the data prepared in the previous determination of the number of distinguishable states in Section 4.4.2. Using deterministic automata a state has to have the same response at the output caused by the same input word which means the achievement of the same following state. If two states differ in their internal bit combination but always respond equally at the outputs then the algorithm identifies these states as only one state. With this, the automaton is not only identified but also reduced. Several states can share the same output word. This is valid for all output words. Therefore, it is possible that all output words of two states are identical without any redundancy. The maximum number of apparently identical states (NoAIS) can be calculated as follows.

$$\max(NoAIS) = \max(NoS) - NoDS \tag{4.14}$$

where NoAIS is equal to the maximum number of states minus the number of securely distinguishable states. For a final distinction of states their state trees are investigated. A state tree contains information of particular output words which are caused by the respective input word applied. As previously described the evaluation of the following output word is an adequate further distinctive feature. Therefore, all following output words (FOWs) of the previous final points are also gathered. This classification is continued until the significance of the trees is sufficient to clearly separate occurring states. Traditional solutions require the knowledge of the maximum number of states [Lee96]. This is an essential disadvantage as the maximum number of states is not available in practice.

However, the restriction to resettable automata or automata with a definable entry point provides the possibility to determine the number of states using an iterative approximation without any knowledge of the real number of states. The more precisely the initial value is predefined the faster and safer the solution of the investigated unknown automata is found out. As previously described the initial value can be either given by the user or is derived from the number of distinguishable states. In this case the number of discriminable states represents the minimum number of states. A predefined number of states is added to this number of distinguishable states. From this predefined number it is expected that many other

similar states exist, which are not distinguishable by only one step. The length of the state trees is calculated as illustrated in Equation (4.15).

$$length_{tree} = NoS - NoDS + 2 \tag{4.15}$$

The added two is based on the matter of fact that two output words form a pair at the rough classification of states. The number of trees (NoTr) with maximum length can be calculated as in (4.16).

$$NoTr\left(maximum\ length\right) = NoS + 1 \tag{4.16}$$

Two states are not redundant if they differ in at least one following state regardless if this is distinguishable from the output word or not. Passing through the complete state trees a difference to all other states will occur due to this condition. Each found state can be retrieved because the path from the initial state to the considered state is saved if it is recognised as a new state. If the state has to be reached again the automaton has to be reset. From the initial state the desired state is retrieved. The state tree is recorded while passing from the current investigated state to the end of the tree. During this process all occurring output words are saved. If the end of the branch is reached the automaton is reset and the next branch is investigated. By applying all possible input words step-by-step a tree is subsequently generated. Each tree consists of a number of branches, where each branch differs in a minimum of one input word. Beginning with the initial state each state is investigated towards its following states. Whenever a new state is found the related output word, the tree and its position relative to its initial state is recorded. For each state and for each input word the following state is determined. The different following output words of the state tree are stored in an array. The required length of the array is calculated as in Equation (4.17).

$$length_{treearray} = \sum_{n=0}^{length_{tree}-1} NoIW^{n} \tag{4.17}$$

Each output word in this array is controlled by a sequence which is stored in another path array. The position of the output word can be determined by the following Equation (4.18).

$$\begin{aligned} position_0 &= 0 \\ position_n &= \left[patharray(n)+1 \right] + position_{n-1} \times NoIW \end{aligned} \tag{4.18}$$

At the position where n is equal to the length of the path array the wanted output word is located. Because of this structure the input words need not to be stored. Therefore, its tree is recorded and compared with all already recorded trees. If the tree is identical then the previously recognised state is entered as following state. Otherwise, a new state is generated and recorded as the following state. As soon as the last found state is investigated towards its last following state the automaton is fully determined. To illustrate the operation of the identification the next section provides an example.

4.5.2 Example of a Mealy Automaton

In this section the main algorithm is exemplified with the following tables and figures. An example of a Mealy automaton as illustrated in Figure 4.6 which is also useful to describe redundant states.

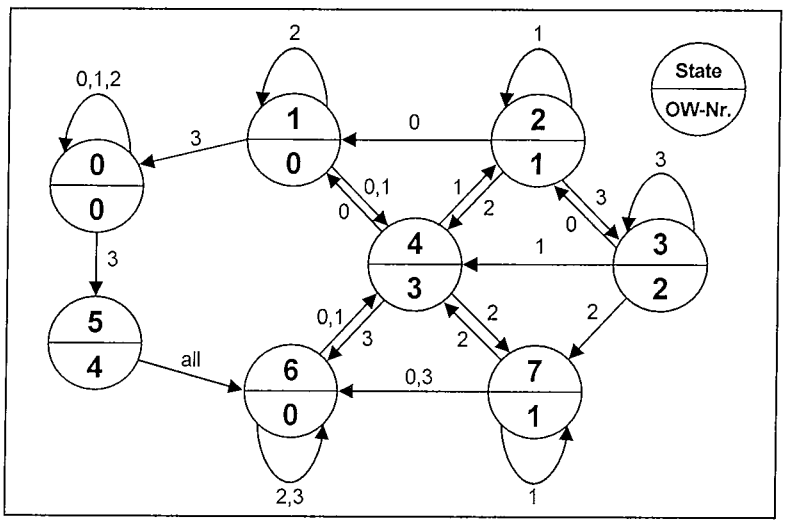

Figure 4.6: State Diagram of a Mealy Automaton

In the top right corner in Figure 4.6 state means the current state and OW-Nr. is the associated output word number. It is an index where the output word combinations are stored. Furthermore, it is a simplification to similarly process Moore and Mealy automata with the presented algorithm. The unknown IC to be investigated has two inputs (NoIP = 2). Therefore, possible input words to cause the state transitions are

'0', '1', '2', '3'. Moreover, the state diagram consists of many states with same output words. However, most of them already differ after one step from the state using each input word. After one step state number '1' as well as state number '6' reacts identically at the output. In Figure 4.6 seven of overall eight states differ either directly or with the following output words after one step. Using this investigation depth the states '1' and '6' appears as one state. The difference between the maximum state number and the number of distinguishable state which is equal to one is extended by two to get the maximum tree length. Here, a preliminary reduction is possible because the states are compared in relation to their current output word as well as their following output word. For a general discrimination two output words are required at time t and at t+1 after one step. In this case the maximum tree length is equal to three. However, the own output words, the following output words and the second following output words are compared. Here, the maximum tree length reveals if either a difference occurs or the states are identical or redundant. Each found state can be reached again, because the path from the initial state to the related state is recorded if the state is identified as new. Table 4.3 clarifies the used structure. The lines with the most significant input words are highlighted in grey.

OW-Nr.(t-1)	IW	OW-Nr.(t)
0	0	0
0	0	3
0	1	0
0	1	3
0	2	0
0	3	0
0	3	4
1	0	0
1	1	1
1	2	3
1	3	0
1	3	2
2	0	1
2	1	3
2	2	1
2	3	2
3	0	0
3	1	1
3	2	1
3	3	0
4	0	0
4	1	0
4	2	0
4	3	0

Table 4.3: Data Structure of the Example

After all entries are made in the table above the information is interpreted. If all of the highlighted lines are added, it will be seen that only seven of eight states can be differentiated. The non distinguishable states are number '1' and number '6' whose input words are both followed by the same output words.

OW-Nr.	OWC				MSIW
0	0	2	1	2	0
1	0	2	3	0	3
2	0	3	0	3	0
3	1	1	1	1	0
4	3	1	0	1	0

Table 4.4: Data Structure of the Example using MSIW

In Table 4.4 the data structure of the example is reduced to the existing most significant input word with the parameters given in Table 4.5.

Type of automaton:	Mealy
NoIW:	4
FNoOW:	5
NoS:	8
NoE:	24
NoDS:	7

Table 4.5: Parameters of the Example

Figure 4.7 illustrates the procedure that leads to the solution. For each state a tree of output words is gathered.

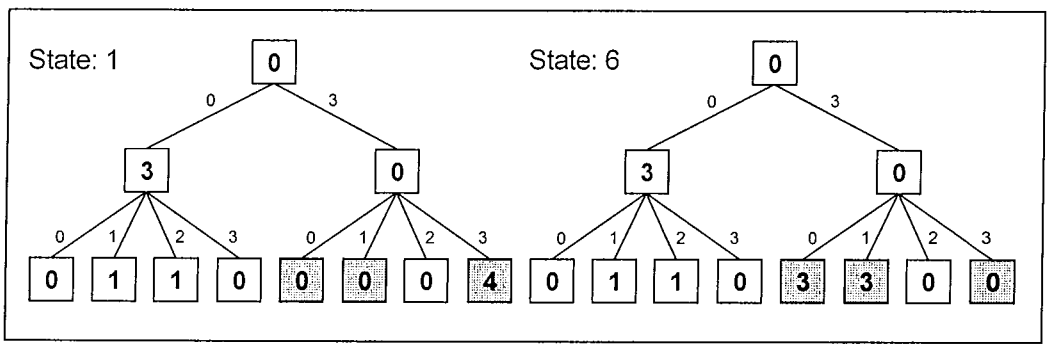

Figure 4.7: State Trees of the Example

The length of this tree is calculated as in Equation (4.19) where NonDS is the number of not distinguishable states.

$$length_{tree} = NonDS + 2 \tag{4.19}$$

The states in these trees will only together cause the same output word if the states which the trees belong to are identical. If they are not then there will be discrepancies in the output words that render it possible to identify the different states. The last part of the example shown in Figure 4.7 does not show the second ($IW = 1$) and third input word ($IW = 2$), since they contain no differences and omitting them provides a better lucidity. The distinguishable entries have been highlighted in grey. Once again the following output words of the last branch of both trees are compared and the discrepancies are found. The gathering of the trees is the main problem. Since it is necessary to return to the initial state of the tree to be able to make the respective steps, there must be a possibility to get back to this state. Therefore, the previously described sequence is used. This will always lead to the same final state. The use of flags and the choice of the input words work in the same way. Likewise, is the test for the completeness of the information. However, the

difference is the amount of states to be checked. The previous procedure tested as many states as existed if they were solved, but this test only checks the information of the number of already distinguished states. The latter might be useful if states are different with respect to their bit combination but still cause the same following states and the same output word. To save processing time every unknown state's output word will be compared with the output words of existing states. If there is no state with this output word yet, then a new state is created and its tree is gathered. The number of this state is entered as the following state of the previous one. Since this state is not at the end of the sequence, another array is needed to enable the re-entry in this state. While the sequence saves output words, this is not necessary there for the path. Since the last state in the sequence is always reached at the beginning of the positioning, it is enough to follow the input words that have been used before arriving at the same state. These input words are stored in path, one for each step that leads away from the base. The length of the array is calculated with Equation (4.20).

$$length_{array} = NoS \times 2 \tag{4.20}$$

Thus, it provides enough range for transitions. If a state is recognised as the first state then the number of steps, which is stored in another variable, will be set to zero. This avoids unnecessary steps and speeds up processing. If the maximal length of the path is reached the algorithm will be reset to zero and it will be continued to determine the automaton from the first state at the end of the period. Thus, the respective last state of the automaton that has been entered can be reached again as often as necessary. The gathering of the trees works analogue to the first tree, but after the end of the sequence the end of the path has to be also found. As mentioned before, output words which are not yet related to known states directly cause the creation of a new state. If the output word is already in use then the corresponding tree is recorded, stored temporarily and compared with all states that share this output word. If one of the existing trees matches then its number will be entered as the following state. In case of a tree that differs to all trees another new state is created, the temporary tree becomes permanent and the number is entered as following state. Since the input words are chosen selectively, the procedure is very effective and the information wanted is gathered quickly. If the number of states

found exceeds the expected number of states the main algorithm is recalled. Then the investigation depth is adapted until the result is within the valid range. The slow identification of unknown automata will be explained now which is a subgroup of the fast identification.

4.5.3 Slow Identification of Unknown Automata

A modification of the fast identification of resettable automata was developed which slows down the whole procedure. In this case the amount of stored values can be reduced dramatically to even carry out the analysis. Using the fast identification each state tree is recorded from its initial state and afterwards compared with other state trees. In contrast, using the slow identification the tree length is reduced as far as the number of maximum states can be stored in the user memory. The shorten state tree cannot unambiguously differ the states. However, with its help a preselection can be made for states which has to be completely investigated branch by branch. The number of trees with short length can be calculated as in (4.21).

$$NoTr\left(short\ length\right) = NoS + 1 \tag{4.21}$$

If there is no difference between the newly recorded state tree and one of the previously stored trees each branch of the currently regarded state is compared with the according one of the apparently conforming state tree. As soon as a difference occurs the algorithm detected that the states are not identical. If two short trees are identical then the state will be compared using their branches. If two states are identical then the state investigated is identical or redundant to the previously state investigated. Generally, after a few investigation steps the differences between two trees occur. The likelihood is very small that two states differ only in the last step or in the last level of the state tree. Therefore, a hybrid solution was developed from the slower identification. It combines the fast analysis comparing the state trees and the slow analysis comparing branch by branch. Considering a large number of different states without recording and comparing the whole state tree this solution provides the execution of the identification algorithm. Therefore, the procedure firstly compares shorter state trees. If two states are not different then their entire trees are compared to detect a potential identicalness. Using the proposed procedure it is

possible to investigate and solve unknown integrated circuits whose states can only be distinguished with large complexity. In addition, requirements to the user memory are reduced without losing all advantages of the default analysis procedure.

After running the main algorithm the two following cases are possible as shown in (4.22), where the NoGS is the number of guessed states and NoFS is the number of found states.

$$\begin{aligned} &\{1\}\ NoFS \leq NoGS \text{ and } NoFS \geq NoDS \\ &\{2\}\ NoFS > NoGS \text{ or } NoFS < NoDS \end{aligned} \tag{4.22}$$

In case {1} the algorithm categorises the result as valid and no further investigation is done. Case {2} reveals that the tree length is too short and has therefore, to be adapted to the number of states found. This has to be done to securely recognise a possible new state. In turn the number of guessed states is equal to the number of found states increased by two. This is repeated until the number of found states is inside the new boundaries. It has to be mentioned that the described procedure has a small error rate, because it depends on the number of found states. If the number of states is too big and differs by a lot of steps from the actual state the iteration procedure possibly draws the false conclusion that it has already detected and discriminated all states. This occurs because the important range to be investigated is out of the range of the search algorithm. However, such cases are only of theoretical interest because the separate states have to achieve these functions. This behaviour can normally be observed at the outputs after a short time. If the real number of states is already known the result can be fully determined without any error. At the end of the analysis procedure the algorithm independently saves the information about the type of the automata (Moore or Mealy), the reset capability, the state transition table, the output function and the simulation time in a text file for further usage. Additionally, in case of a detected Mealy automaton the output word list is stored containing the allocation of the output word combinations to the states. After having provided the slow solution of nonlinear unknown ICs the next section will now explain the determination of an entry point for non-resettable automata.

4.5.4 Determination of an Entry Point for Non-Resettable Automata

A distinctive feature of automata is the reset capability. If such an active reset capability is not available for the analysis a well-defined entry point has to be found which provides the analysis algorithm with an initial point or root point. This initial state must be located at the end of a path which is unambiguously described by its length and by its passed output words. Therefore, the real number of states has to be known. The search can be started at a random point. For this, at each respective state the most significant input word is applied. Thus, it has to be ensured that as many states as possible are reached during this search. However, two constraints have to be fulfilled to find the initial state. First, the length of the path recorded has to be sufficient. Second, it must be ensured that this path investigated is not at the end of its second cycle. Analogous to the determination of tree length the distance depends on the difference of safely distinguishable states and the given maximum number of states. In this case reset means the search of an output word which is identical to the output word of the path found in the beginning and tracing this path to its end. If there is a difference between the expected and the real values then the first output word of the path is searched again. This is done until no deviation occurs and therefore, the initial state is reached. If the entry point is found the analysis procedure can be carried out analogous to fast or slow identification of unknown resettable IC as was previously described. To find the initial state the real number of states of the automaton under investigation has to be determined. After that, any state is chosen and many clock cycles are run through which are equal to the double tree length. Using the real number of states the tree length is calculated as in Equation (4.23)

$$length_{testpath} = 2 \times (NoS - NoDS + 2) \tag{4.23}$$

where most significant input word is applied to the automaton which corresponds to the actual output word. Each output word which is found during the preparing algorithm is allocated to a most significant input word. Therefore, for each output word of the unknown automaton occurring one most significant input word is available. In the next step one state will be determined as the initial state with following properties. The previously recorded output words must unambiguously

conclude that up to the middle of the sequence recorded, a sequence exists which is unique for the overall automaton. The middle of the sequence recorded characterises the first half of the test path which forms the entry path. The entry path can be considered as unique if the following condition holds. After the analysis of the overall test path it must be excluded that reaching the last output word of the sequence the automaton stands in a cycle greater than one.

Most of today's integrated circuits already have an initial state, which can be reached directly using a reset pin or by means of disconnecting and reconnecting the supply voltage. Therefore, the main focus is on the resettable types of automata. All resettable types of automata can be solved using the described algorithm, because they provide the needed entry point automatically. In the next section a modification of the investigation depth will be described which is suitable for hardware analysis.

4.6 Summary

This chapter has presented a novel non-destructive algorithm to identify unknown nonlinear ICs. In Section 4.1 a general overview of related work was given. Section 4.2 then discussed the structure of the algorithm which consists of three main parts. The separation into Moore or Mealy automaton is the first part of the overall algorithm and was described in detail in Section 4.3. It is used to choose the further procedure which provides the best efficiency. In Section 4.4 the preparation algorithm was given which is the second part of the algorithm described. The preparation consist of four main steps which were described in detail. However, in practice the number of states is not always known. Therefore, a novel iteration procedure was introduced by which the algorithm independently approximates to the real number of states if necessary. Here, the determination of the most significant input word as well as and the calculation of the number of distinguishable states are the important parts to prepare the main algorithm. This was presented in Section 4.5 and is the third and most complex part of the overall analysis. In Section 4.5.1 the identification of nonlinear ICs was described. To demonstrate the operation an example was explained in Section 4.5.2. It uses an optimisation to detect redundant states. Furthermore, in Section 4.5.3 the slow identification was introduced. It is a modification of the fast identification which reduces the amount of storage. Hence,

the algorithm can be used to identify complex ICs. Normally, the ICs under investigation have a reset capability. If the IC is not resettable Section 4.5.4 then presented an approach to find an entry point for non-resettable automata. However, a solution cannot be predicted because a unique entry point for the algorithm has to be found. Nevertheless, the described algorithm tries to find a solution. In the next chapter the analysis environment will be presented. Furthermore, ISCAS models of unknown ICs will be introduced to demonstrate the proper operation of the introduced algorithm. Finally, and the results obtained will be presented and evaluated.

5 Results

The previous two chapters have shown novel procedures to analyse unknown ICs. In Chapter 3 the determination of pin types as well as the separation procedure to determine the independent bocks of an IC was explained, while the identification of nonlinear unknown ICs was introduced in Chapter 4. This chapter now first presents the analysis environment to simulate and analyse unknown integrated circuits in a non-destructive manner. Then, in Section 5.2 the IEEE International Symposium on Circuits and Systems (ISCAS) benchmarks of the series 1985, 1989 and 1999 are introduced. These models of unknown ICs are well suited for test and analysis. After this, the implementation into the analysis environment is described using different examples. In Section 5.3 the ISCAS benchmarks as well as several user defined models are used to show the correct function of the separation procedure as was explained in Chapter 3. Finally, the results obtained from these tests and simulations are presented. Furthermore, the correct operation of the analysis of nonlinear FSMs described in Chapter 4 is verified in Section 5.4.

5.1 Analysis Environment

The procedures presented in Chapter 3 as well as in Chapter 4 have shown the need for an analysis tool. To achieve this, an analysis environment was developed and implemented to verify the correctness of the algorithms developed, both for simulated as well as for real integrated circuits. Therefore, the general structure of the analysis approach developed is described which can be applied to the analysis of finite state machines in unknown ICs. However, the analytic process of FSMs in CMOS ICs requires only short processing times. In contrast, the identification of specific subgroups and the subsequent analysis of mathematical behaviour is a time consuming process. Therefore, the analysis environment is divided into two parts, the user interface (UI) and the development board as shown in Figure 5.1.

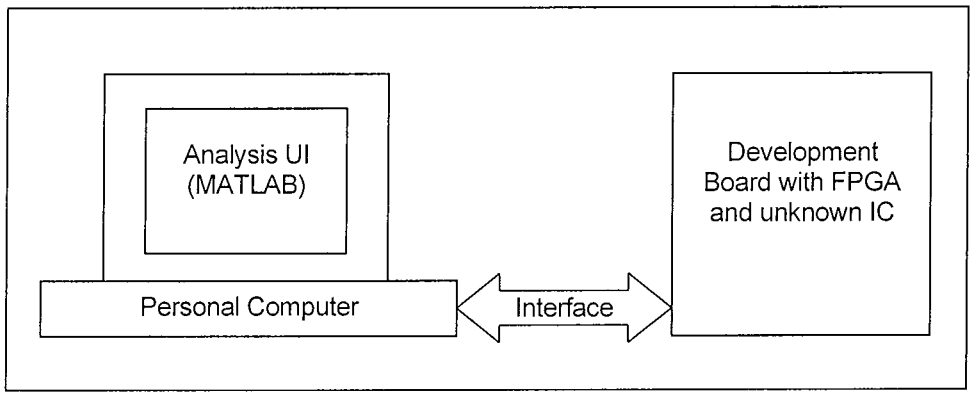

Figure 5.1: Principal Structure of Analysis Environment

As can be seen in Figure 5.1 the analysis environment consists of a personal computer (PC) and a field programmable gate array (FPGA). Both systems work together to execute a real analytic process at an unknown CMOS integrated circuit. The PC used has a Dual Core CPU operating at 1,87GHz together with 2GByte of RAM. The user interface was realised using MATLAB R2008b [Matl08] and is used for a priori determination of information which are needed for the analytic process. This information will be specified at the beginning of the analysis process. Then, this information is transmitted to the FPGA. After having described the general structure of the analysis environment the user interface as well as the development board will be described in detail in the next sections.

5.1.1 User Interface

The user interface is the first part of the analysis environment. The main feature of this interface is the creation of input values as well as the evaluation of output values. In addition, several parameters are defined which are needed for further analysis. Figure 5.2 shows the structure of the analysis interface in general. Both the simulation and the hardware analysis work similarly.

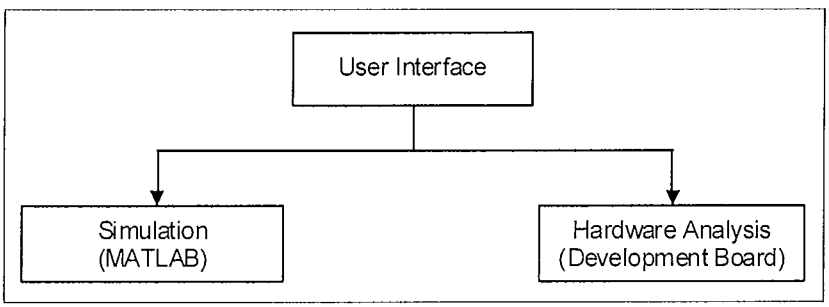

Figure 5.2: Structure of the Analysis GUI

When choosing simulation the algorithm investigated can be applied to a virtual model of an integrated circuit. If this test works correctly then the hardware analysis can be selected. Here, the same algorithm will be applied to a real integrated circuit connected through the development board. However, the analysis user interface controls the overall analysis. In the next section the development board will be given which interfaces the PC with the unknown IC.

5.1.2 Development Board

The development board is the second part of the overall analysis environment. It consists of an FPGA and the hardware which is needed to analyse an unknown ICs. Additionally, the combinations of input values are transmitted through an interface from a PC to the analysis board which is also a part of the development board. However, an FPGA is a freely programmable semiconductor and has the advantage that it can be used to program the interface as well as the test environment. The evaluation board must fulfil the requirements needed for the implementation and therefore, a FPGA from the company ALTERA [Alte09] with sufficient hardware was chosen. Figure 5.3 shows the development board DE2-70 which was used in this research.

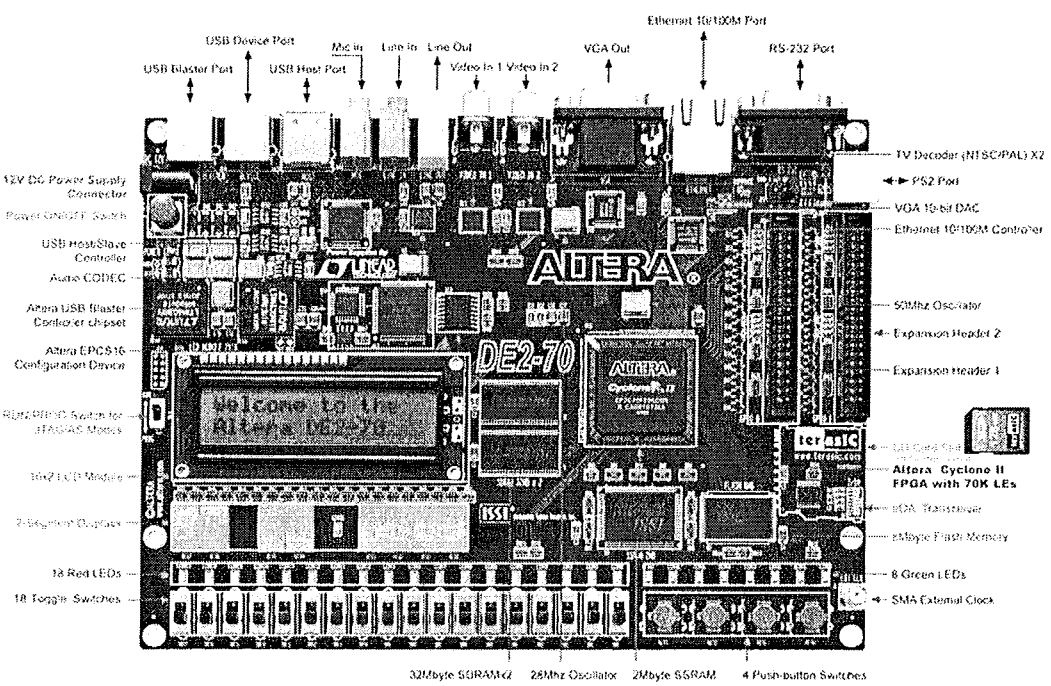

Figure 5.3: Development Board DE2-70 [Tera09]

The ALTERA Cyclone II FPGA 2C70 with 70000 logic elements (LEs) is the core of the development board. It can be either clocked by different onboard oscillators or by external clocks. In this research the 50MHz onboard oscillator is used. Hence, the two 40-pin expansion header as well as other connections can be used to connect the unknown IC to be investigated. Furthermore, the onboard RS-232 transceiver is also used in this research. To program the FPGA ALTERA provides a software environment called Quartus II [Alte09]. After describing the development board in general an overview of modules implemented into hardware will be given. It includes the interface with receiver, transmitter, decoder, coder and interface to the unknown integrated circuit as illustrated in Figure 5.4.

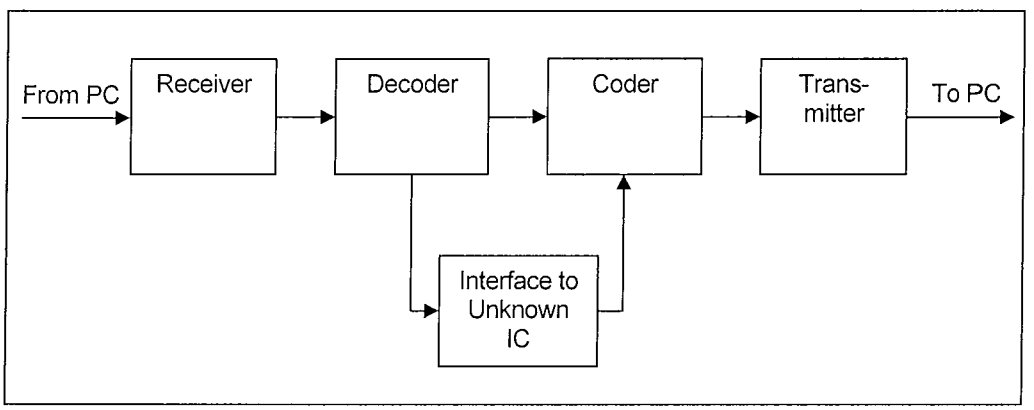

Figure 5.4: Overview of Modules Implemented

The receiver is the first part of the modules implemented and will be explained in the next section.

Asynchronous Receiver

The receiving module identifies and edits the incoming data packets. Afterwards, it transfers the data to the next module. Therefore, the module is subdivided into different structures which execute the several tasks. For the correct mode of operation of the receiving block a synchronisation between the input sequence and the converting of data must be carried out. This means, that the module must process data with the same frequency as the receiving data sequence. As explained in the previous section the onboard frequency generator is used which has a clock frequency of 50MHz. From this frequency the baud rate of 115200Baud is derived. The receiver module implemented is illustrated in Figure 5.5.

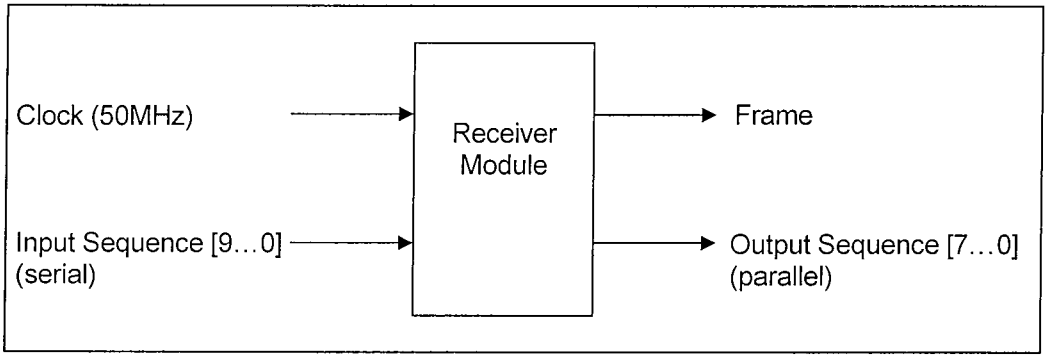

Figure 5.5: Implemented Asynchronous Receiver

As can be seen in Figure 5.5 several input and output parameters are illustrated. Again, the input sequence has a baud rate of 115200Baud where the data packets contain 10 bits. Here, 8 bits are data and 2 bits are additional information which is the start bit and the stop bit. They are used to identify the data and transfer the data with an output sequence to the next module. Since, the output sequence contains 8 bits and the transmission is carried out permanently an identification of 8 bits related must be executed. Therefore, an additional signal is transferred which signalises the first and the last bit of the 8 bits frame. Here, the identification uses a frame signal which has the value '1' if the data sequence starts and it has the value '0' after the data has been transmitted. In the following section the mode of operation of the decoder is described.

Decoder

The decoder is the main unit implemented into the FPGA. It has two input signals: the frame signal and the input sequence. The frame signal is used for the identification and synchronisation of the input sequence. This means, that the decoder recognises one input sequence with data during the frame signal is '1'. The decoder is illustrated in Figure 5.6 showing all possible inputs and outputs.

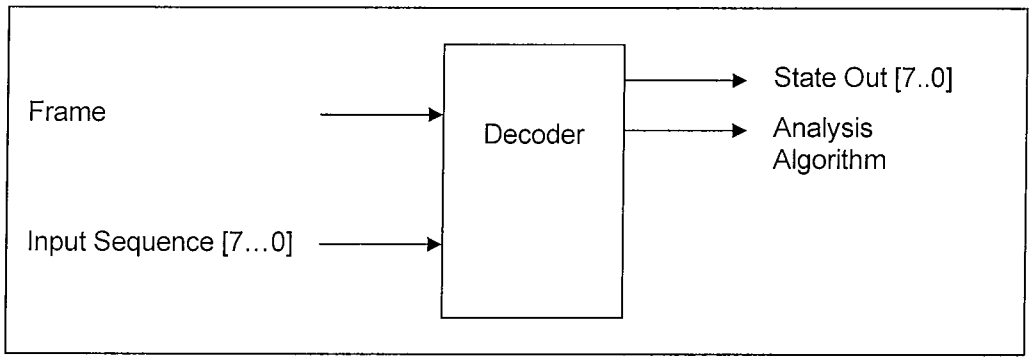

Figure 5.6: Implemented Decoder

The decoder is directly connected to the interface module to the unknown IC as well as to the coder. The incoming data sequence contains different information. Based on this information the useful data are transferred to the according module. The pin sequence contains the pin types and pin numbers. Next to the pin information the input sequence is received in the decoder which is used to analyse the unknown IC. In the next section the interface to the unknown IC will be given.

Interface to Unknown Integrated Circuit

The interface to connect the integrated circuit to the hardware board has two main functions. First, it assigns the pin types. Secondly, the interface applies the input sequence. Furthermore, the interface is connected to the decoder as well as the coder. Figure 5.7 shows the inputs and outputs of the interface module implemented.

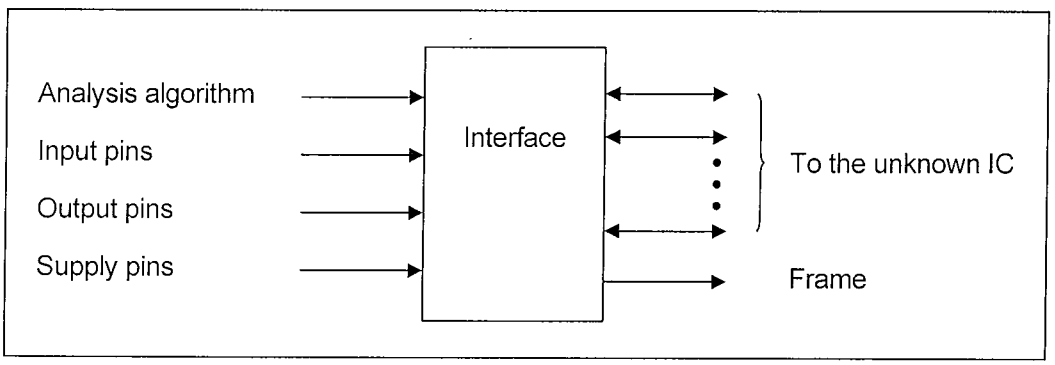

Figure 5.7: Implemented Interface to Unknown IC

After applying the input sequence to the unknown IC the output values are recorded and transmitted to the coder. To test the overall procedure the IC under investigation can be also implemented into the interface which is also called internal analysis. For

real analysis Figure 5.8 shows the experimental setup using the development board DE2-70 and an unknown IC.

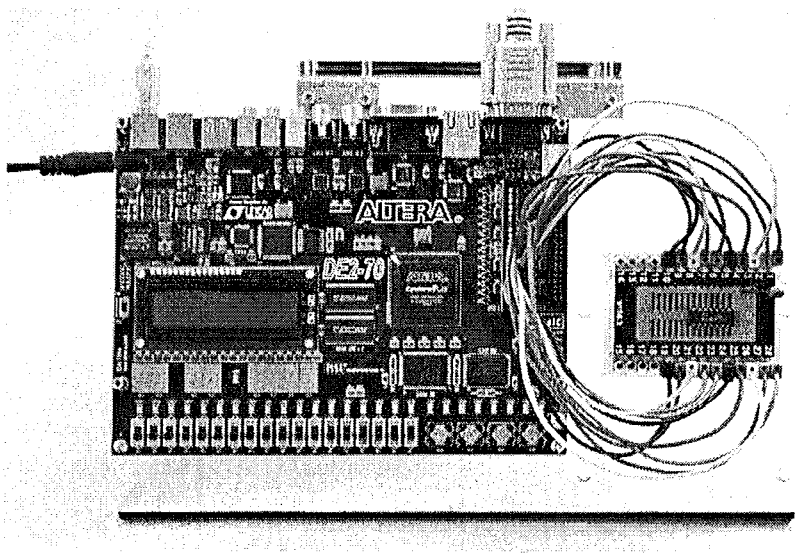

Figure 5.8: Development Board Analysing an Unknown IC

Using either internal or real analysis the timing behaviour of the IC to be investigated is nearly identical. Therefore, to simplify the simulation predominantly internal models were used for the analysis. The coder will be explained in the next section.

Coder

The coder adds control character to the results obtained. The coder module implemented is illustrated in Figure 5.9.

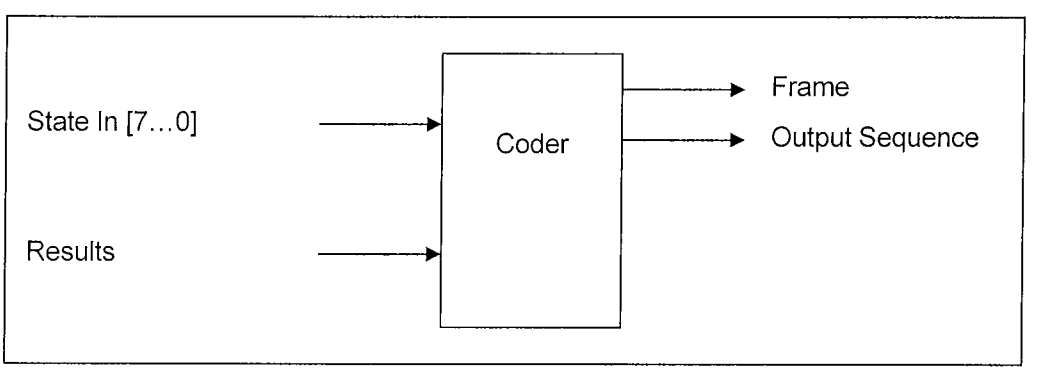

Figure 5.9: Implemented Coder

The results are packed in a data stream and afterwards, they are transmitted to the transmission module. In the following section the transmission module will be described which will be used to send the information.

Asynchronous Transmitter

The transmission module receives the results of the analysis from interface. Furthermore, the data are edited and adapted to transmit the data to PC. This adaptation is necessary, because the transmission of data is executed on the basis of the RS232 protocol. Therefore, the transmission module has the task to convert the data stream received in data packets of 8 bit. Furthermore, the data are marked with one start and one stop bit. The transmission module is illustrated in Figure 5.10.

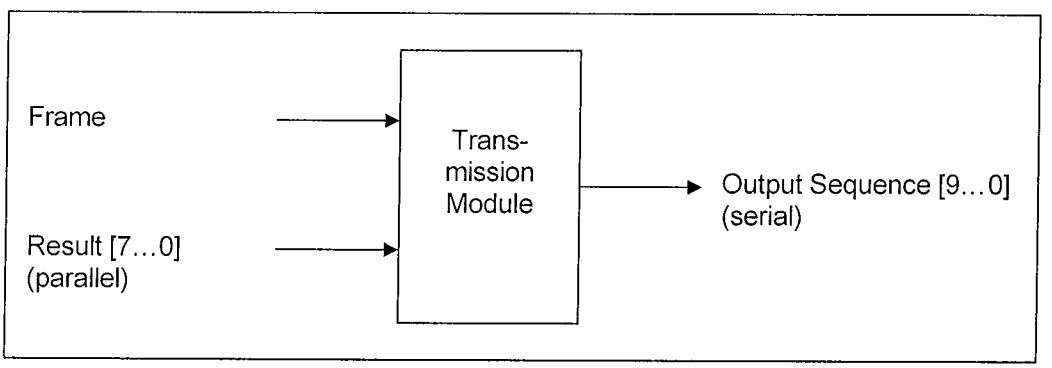

Figure 5.10: Implemented Asynchronous Transmitter

It shows all available inputs and the serial output. At the input result the transmission module receives the output values of the unknown integrated circuit, which corresponds to the results of the analysis. Furthermore, an input exists which is denoted as a frame and has the task of identifying the start and the end of a data packed of 8 bits. These bits are identified as data which is transferred to the PC. Here, the results will be evaluated. The transmission module is connected to the personal computer by an interface, which sends the data by means of the RS232 protocol. After describing the analysis environment in detail the data transfer between PC and development board will be explained in the next section.

5.1.3 Data Transfer Protocol

The data which are transferred between PC and development board must ambiguous. Therefore, a standardised format is used. Depending on the data to be sent both the

PC and the development are able to evaluate the information. Here, all information streams are identified with ASCII signs. The pin numbers and the pin types are declared with three ASCII signs 'p', 'i' and 'n' while the algorithm applied is identified with two ASCII signs 'i' and 'n'. With the help of this sign the evaluation board and the user interface can identify unambiguously the subsequent information sequence. The pin information is sent as illustrated in Figure 5.11.

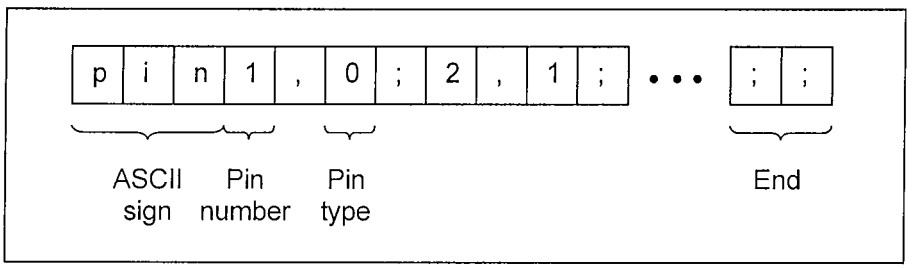

Figure 5.11: Example of Data Stream for Pin Numbers and Pin Types

After transferring the information as illustrated in Figure 5.11 the algorithm is applied to the unknown integrated circuit. However, the output values are transmitted backward to the user interface. The whole data sequence of the algorithm is demonstrated in Figure 5.12.

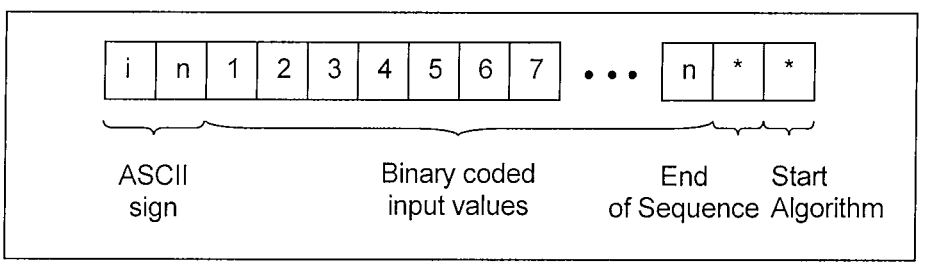

Figure 5.12: Example of the Transmission of Algorithm

The binary coded input values are used as follows. Field '1' contains the input values for the first 8 inputs (0 to 7). Field '2' contains the input values for the next 8 inputs (8 to 15) and so on. As can be seen in Figure 5.12 the number of fields can be easily extended. At the moment eight fields are implemented which results in an input vector of 64 possible inputs. The same bit width of 64 is implemented as possible outputs. This transmission procedure is executed until the transmission of the algorithm is ended by two asterisks. The end of the sequence is labelled by the first asterisk. The second asterisk signalises the start of the algorithm applied. However, it is possible to apply the combination of input values and to record the combination of output values during the same procedure. In the next section IC

models are introduced which were used to verify the correct functioning of the analysis developed.

5.2 IC Models for Analysis

In the area of digital circuit testing the IEEE ISCAS, which meets every year in May, set the standards for the design for testability (DFT) work with the ISCAS-85, ISCAS-89 and ISCAS-99 benchmark circuits [Hans09]. These circuits are industrial designs whose functions and high-level designs have not been published, both for confidentiality reasons and to allow them to be viewed as random logic circuits with no significant high-level structure. These benchmarks were introduced to investigate combinational and sequential automatic test pattern generation (ATPG) tools [Kim05]. These benchmark circuits are now used in a wide variety of applications in the microelectronic circuit test field, as well as to evaluate the performance of new methods in other areas, including logic optimisation, power estimation and partitioning [Corno00]. The ISCAS circuits became important vehicles for industry and academics to develop new tools, compare and contrast different methodologies, research new algorithms and techniques and fault diagnosis [Frie71]. Therefore, the ISCAS benchmarks represent an appropriate target for reverse engineering. Furthermore, they are suitable to demonstrate the function as well as the performance of the non-invasive procedure. This section refers briefly to the reverse engineering work of Hansen, Yalcin and Hayes which was carried out at the University of Michigan [Hans99].

The ISCAS-85 benchmarks can be described with the help of high level models. These benchmarks are nonlinear combinatorial FSMs with only one state. However, despite of this exception they are suitable to proof the correctness of the introduced algorithm. The ISCAS-89 benchmarks are nonlinear FSMs of both Moore and Mealy. Furthermore, the models used were slightly modified and a reset capability was added to models without an already existing reset. The ISCAS-99 benchmarks have been developed in 1999. These benchmarks contain also nonlinear automata both of type Moore or Mealy. Furthermore, the ISCAS-99 models need not to be changed, because the already have a reset pin. The following Table 5.1 shows an overview of all analysed ISCAS-85, ISCAS-89 and ISCAS-99 benchmark series

with the most important characteristic and a short function conclusion if available. A detailed discussion of these circuits can be found in [Hans99], [Isca08], [Muel08a], [Muel08b] and [Hans09].

Circuit Name	Ins	Outs	Gates	Flip-Flops	FSM Type	ISCAS Series	Circuit Function / Behaviour
C17	5	2	6	0	Mealy	1985	Combinatorial
S27	4	1	11	3	Mealy	1989	Nonlinear Seq.
S298	3	6	160	15	Moore	1989	Nonlinear Seq.
B01	2	2	49	5	Moore	1999	FSM that compares serial flows
B02	1	1	28	4	Moore	1999	FSM that recognises BCD numbers
B03	4	4	160	30	Moore	1999	Resource arbiter
B06	2	6	56	9	Moore	1999	Interrupt handler
B11	7	6	770	31	Moore	1999	Scramble string with variable cipher
EC1	2	3	3	0	Mealy	user	Combinatorial
EC2	6	4	5	0	Mealy	user	Combinatorial
EC3	8	6	9	0	Mealy	user	Combinatorial
ELS1	3	2	7	3	Mealy	user	Linear Sequential
ELS2	6	4	13	6	Mealy	user	Linear Sequential
ELS3	9	7	19	10	Mealy	user	Linear Sequential
ELS4	10	8	23	13	Moore	user	Linear Sequential
ENLS1	3	2	7	3	Mealy	user	Nonlinear Seq.
ENLS2	6	4	13	6	Mealy	user	Nonlinear Seq.
ENLS3	9	7	19	10	Mealy	user	Nonlinear Seq.
ENLS4	10	8	23	13	Moore	user	Nonlinear Seq.

Table 5.1: IC Models for Analysis

In addition, self generated examples of several automata are entered in the lower part of Table 5.1. In the left column the single circuits are listed with the attached number of inputs and outputs right of it. The sum of gates can be seen in the fourth column which increases with corresponding more complex function seen on the right. Further information are the number of flip-flops, the type of the investigated FSM and if used the appropriate ISCAS series. The ISCAS models are available as behavioural models in form of VHDL or Verilog Code. In contrast to the benchmarks the user defined models are generated in SIMULINK. However, using the design software Quartus-II the VHDL or Verilog code from each model was chosen, analysed and synthesised. Afterwards, the models of the unknown ICs were

imported into the analysis environment. Moreover, it is possible to edit the models implemented by adding additional functions. After introducing various benchmark models as well as user defined models the results of the separation procedure presented in Chapter 3 are given in next section.

5.3 Separation Procedure

In this section the separation procedure is applied to the benchmarks and user defined models introduced in Section 5.2. Therefore, the analysis steps were implemented into the analysis environment. As explained in Section 3.3 the overall separation procedure starts with the determination of independent blocks which will be firstly described in the next section using the example C17.

5.3.1 Example of the Combinatorial FSM C17

The determination of independent blocks is the first step of the separation procedure which was described in detail in Section 3.3.1. To demonstrate the correct mode of operation the ISCAS-85 model C17 is used which is illustrated in Figure 5.13.

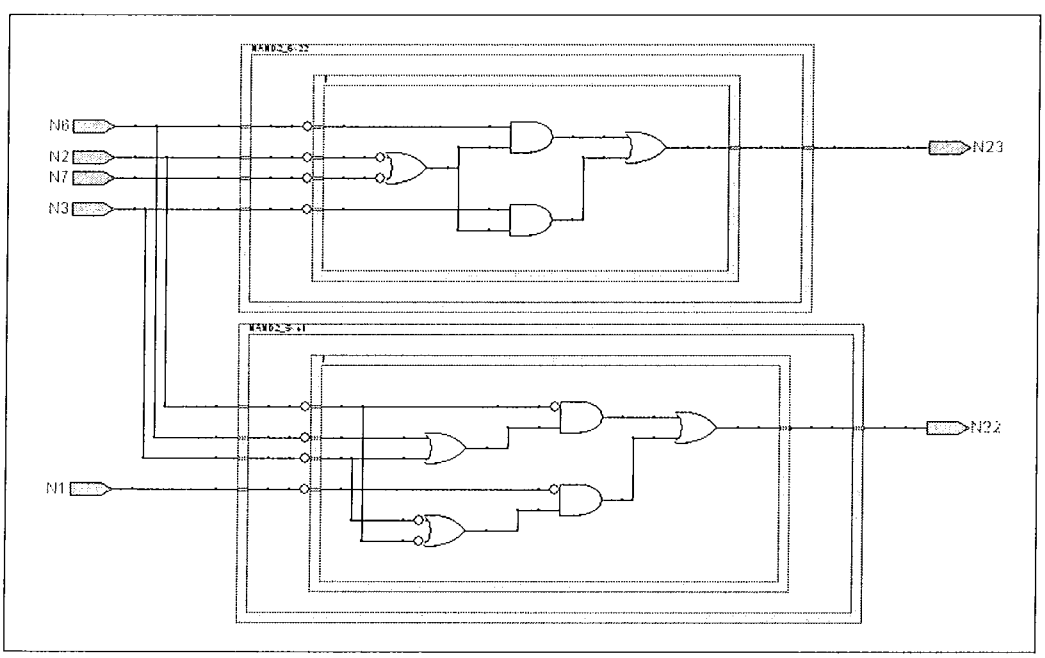

Figure 5.13: The Implemented Model C17

As can be seen in Figure 5.13 C17 model C17 consists of five inputs (N1, N2, N3, N6, N7) and two outputs (N22, N23). Furthermore, the overall IC model consists of

two independent blocks which both have combinatorial behaviour. Each block again consists of five inputs and one output which is not visible to the analysis. However, according to the number of input pins of C17, five pn-sequences are used as described in Table 3.3. They are applied step by step to the model and the outputs are recorded in relation to their inputs. Table 5.2 shows the result array of the determination of independent blocks for the model C17.

	N1	**N2**	**N3**	**N6**	**N7**
N22	1	1	1	1	0
N23	0	1	1	1	1

Table 5.2: Result Array of Independent Blocks of C17

Here, the '1' means that there is a functional connection between the input and the related output. Otherwise, if a '0' exist there is no functional connection between the input the output. The mathematical interpretation of Table 5.2 is given in (5.1).

$$\begin{aligned} C17_1 &= f(N22, N1, N2, N3, N6) \\ C17_2 &= f(N23, N2, N3, N6, N7) \end{aligned} \tag{5.1}$$

Using the separation procedure described two independent blocks were found which conforms to the real hardware. The first circuit block $C17_1$ is connected between inputs N1, N2, N3, N6 and output N22. Internal circuit $C17_2$ is connected between inputs N2, N3, N6, N7 and output N23. The result extracted from the separation procedure is equivalent to the example of the integrated circuit shown in Figure 5.13. After the determination of the independent blocks the division into combinatorial or sequential FSMs will be performed.

The division into combinatorial or sequential finite state machines was described in detail in Section 3.3.2. This procedure is now applied to C17. Here, the same pn-sequences as in the previous section are used for the division into combinatorial or sequential finite state machines. Now, no combinations of pn-sequences but many completely different pn-sequences are applied to all four inputs of each independent block $C17_1$ and $C17_2$ at a time. After applying the pn-sequences the results of the output pins as well as the values of the input pins of the corresponding time steps are stored in a data table. The data table stored of the independent block $C17_1$ is shown in Table 5.3.

Time Index t	Inputs				Output
	N1	N2	N3	N6	N22
1	1	0	1	0	1
2	0	1	0	0	1
3	**1**	**0**	**0**	**0**	**0**
4	1	1	0	1	1
...					
125	1	1	0	0	1
126	**1**	**0**	**0**	**0**	**0**
127	0	1	1	0	1
...					
409	1	0	0	1	0
410	**1**	**0**	**0**	**0**	**0**
411	0	1	0	0	1
...					

Table 5.3: Data Table Recorded of $C17_1$

The second independent block $C17_2$ is illustrated in Table 5.4.

Time Index t	Inputs				Output
	N2	N3	N6	N7	N23
1	1	0	1	0	1
2	0	1	0	0	0
3	**0**	**1**	**0**	**1**	**1**
4	1	0	1	0	1
...					
212	1	1	0	0	1
213	**0**	**1**	**0**	**1**	**1**
214	0	1	1	0	0
...					
301	1	0	0	1	1
302	**0**	**1**	**0**	**1**	**1**
303	0	1	0	0	0
...					

Table 5.4: Data Table Recorded of $C17_2$

Using the theory of combinatorial logic as explained in Section 3.3.2 the results of Table 5.3 as well as Table 5.4 are checked for each time step *t*. Here, the same input sequences must always lead to the same output results. It can be seen from Table 5.3 that for lines 3, 126 and 410 all input values result in '1'. Therefore, it can be concluded that independent block $C17_1$ has a combinatorial behaviour. In Table 5.4 the lines 3, 213 and 302 have the same input combination and therefore, must lead to the same output results for a combinatorial system. This is true and therefore, independent block $C17_2$ has also a combinatorial behaviour. The classification

procedure is now complete. Only in case of sequential behaviour a further classification into linear or nonlinear automata needs to be carry out. As previously described the results of the separation procedure exist in form of arrays and vectors of binary values. However, for clarity the results have to be converted into a human readable form. The result screen as described in (5.2) contains the simulated results for the IC C17 shown in Figure 5.13.

$$\begin{aligned} C17_1 &= f(N22, N1, N2, N3, N6, combinatorial) \\ C17_2 &= f(N23, N2, N3, N6, N7, combinatorial) \end{aligned} \tag{5.2}$$

The following information can be concluded from (5.2). There exist two independent blocks. Block $C17_1$ is of combinatorial type and is interconnected between inputs N1, N2, N3, N6 and output N22. Internal circuit $C17_2$ has also combinatorial behaviour and is connected between inputs N2, N3, N6, N7 and output N23. This result extracted from the separation procedure is equivalent to the IC model C17 under investigation. The whole simulation, preparation, recording of the simulation results as well as the analysis of the results is executed by functions implemented in the analysis environment. After having described the separation procedure using IC model C17 various IC models will be investigated in the next section.

5.3.2 Simulation of IC Models

To demonstrate the correct operation of the separation procedure several IC models will be evaluated in this section. Therefore, IC models with different behaviour as well as different number of inputs and outputs were chosen. Furthermore, the models implemented consist of different structures. Table 5.5 shows the results of IC models implemented and evaluated by the analysis environment. Here, NoIB means the number of independent blocks. Moreover, it has to be noted that C stands for combinatorial, LS for linear sequential and NLS for nonlinear sequential, respectively.

Benchmark	Total NoIB	NoIB Found	Behaviour	Behaviour Found	Evaluation Time [s]
C17	2	2	2x C	2x C	14,6
S27	1	1	1x NLS	1x NLS	4,3
S298	1	1	1x NLS	1x NLS	19,6
B01	1	1	1x NLS	1x NLS	3,91
B02	1	1	1x NLS	1x NLS	1,14
B03	1	1	1x NLS	1x NLS	24,6
B06	1	1	1x NLS	1x NLS	7,4
B11	1	1	1x NLS	1x NLS	169,7
EC1	1	1	1x C	1x C	12,7
EC2	4	4	4x C	4x C	25,6
EC3	4	4	4x C	4x C	88,1
ELS1	1	1	1x LS	1x LS	1,9
ELS2	2	2	2x LS	2x LS	23,2
ELS3	4	4	4x LS	4x LS	240,6
ELS4	5	5	5x LS	5x LS	905,0
ENLS1	1	1	1x NLS	1x NLS	2,0
ENLS2	2	2	2x NLS	2x NLS	22,7
ENLS3	4	4	4x NLS	4x NLS	241,0
ENLS4	5	5	5x NLS	5x NLS	621,3

Table 5.5: Result Table of Benchmarks Analysed

As can be seen in Table 5.5 in addition to C17 also other ICs may consist of several independent blocks, for example EC2, EC3, ELS2, ELS3, ELS4, ENLS2, ENLS3, or ENLS4. However, using the novel separation procedure it was shown that all benchmark circuits under investigation were successfully analysed in relation to their independent blocks as well as their behaviour. As previously described it is also possible that real ICs consist of more than one independent block with the same behaviour. To make the analysis more complex and realistic several benchmark as well as user defined models were combined and tested at the same time. Afterwards, they were implemented into the analysis environment. Here, the independent blocks have different behaviour. Figure 5.14 shows an example of a benchmark mix where three benchmarks with two different behaviours were implemented.

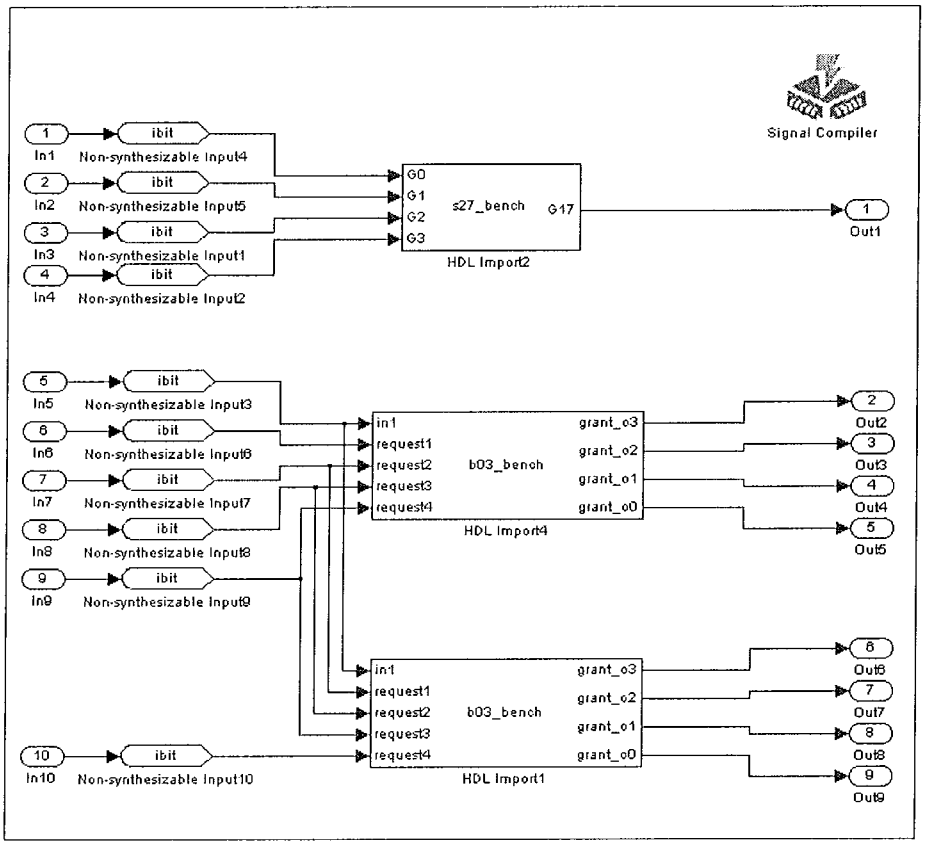

Figure 5.14: Mixed Benchmarks B03 and B06

As can be seen in Figure 5.14 the ISCAS-99 model B03 and two models B06 were simultaneously implemented into the analysis environment. Furthermore, this figure shows the principle of an overall unknown IC which contains subcomponents. These components may consist of VHDL or Verilog Code. However, before starting the separation procedure there is no knowledge about the internal behaviour. Table 5.6 shows the overall results of all circuit combinations investigated.

Benchmark-Mix	Total NoIB	NoIB Found	Behaviour	Behaviour Found	Evaluation Time [s]
2xC17	4	4	C17: 2x C	2x C	582,9
C17, S27	3	3	C17: 2x C S27: 1x NLS	2x C 1x NLS	201,2
2xS27	2	2	S27: 1x NLS	1x NLS	78,6
S27, 2xB03	3	3	S27: 1x NLS B03: 2x NLS	3x NLS	984,6
B01, 2xB03	3	3	B01: 1x NLS B03: 2x NLS	3x NLS	971,1
B01, B06	2	2	B01: 1x NLS B06: 1x NLS	2x NLS	34,8
B03, B06	2	2	B03: 1x NLS B06: 1x NLS	2x NLS	180,8
B01, B11	2	2	B01: 1x NLS B11: 1x NLS	2x NLS	854,2
B03, B11	2	2	B03: 1x NLS B11: 1x NLS	2x NLS	1070,9
B01, B02, B06	3	3	B01: 1x NLS B02: 1x NLS B06: 1x NLS	3x NLS	153,1
EC1, ELS1, ENLS1	3	3	EC1: 1x C ELS1: 1x LS ENLS1: 1x NLS	1x C 1x LS 1x NLS	78,9
EC2, ENLS1, ENLS2	7	7	EC2: 4x C ENLS1: 1x NLS ENLS2: 2x NLS	4x C 3x NLS	991,3
ELS1, ELS4	6	6	ELS1:1x LS ELS4: 5x LS	6x LS	1099,3
ENLS1, ENLS4	6	6	ELS1: 1x NLS ENLS4: 5x NLS	6x NLS	942,6
EC1, ENLS3	5	5	EC1: 1x C ENLS3: 4x NLS	1x C 4x NLS	987,2
ELS1, ENLS4	6	6	ELS1: 1x NLS ENLS4: 5x NLS	6x NLS	816,4
EC1, ELS3	5	5	EC1: 1x C ELS3: 4x LS	1x C 4x LS	1007,0
EC3, ELS2	6	6	EC3: 4x C ELS2: 2x LS	4x C 2x LS	1042,0

Table 5.6: Result Table of Mixed Benchmarks Analysed

From Table 5.6 two important information can be extracted. First, the number of independent blocks was successfully found for all ICs. Moreover, also the behaviour was always correctly identified. Furthermore, the evaluation time in the right column shows that the analysis is accomplished within less than an hour for even very

complex circuits. After having presented the results of the classification procedure the results of the nonlinear analysis will be given in the next section.

5.4 Nonlinear Analysis

In this section the nonlinear analysis as described in Chapter 4 is applied to the benchmarks and user defined models introduced in Section 5.2. Before the nonlinear algorithm can be started, some analysis settings have to be done which will be described in the next section.

5.4.1 Analysis Settings

The adjustment of the simulation mode is the first setting of the nonlinear analysis procedure. In contrast to the separation procedure the nonlinear analysis can be used both to simulate IC models and to analyse real hardware. Here, real hardware analysis means that the unknown IC exists as real hardware or is implemented into the FPGA hardware. Furthermore, the serial port has to be initialised to activate the communication between the user interface and the analysis board. Here, the baud rate of the serial port is set to 115200Baud using one start bit, one stop bit, eight data bits and no parity check. If simulation models are used then the output function and state table of an automaton is already known. This mode is useful to check the correct operation of the algorithm and to verify the results of the solution using an existing output and state transition function. After having described the simulation mode the input-output settings will be given. At the moment a maximum number of 64 input and 64 output pins is implemented for a description of an unknown IC under investigation. Therefore, the number of input pins (NoIP) is set to NoIP = 64 and the number of output pins (NoOP) is set to NoOP = 64. Due to the well organised implemented interface between the user interface and the analysis board it is no problem to extend the number of inputs and outputs. In addition, the arrays for the input pins (IP) and the output pins (OP) have to be prepared. An example for an array of IP with four inputs is illustrated in (5.3).

$$IP = \left[\underbrace{\overset{Clk}{\overset{\frown}{0}}\ \overset{Rst}{\overset{\frown}{0}}\ 110000}_{1.Byte}\ \underbrace{10010000}_{2.Byte} \ldots \underbrace{00000000}_{8.Byte} \right] \tag{5.3}$$

The array IP consists of eight bytes which consist of 64 bits. If the polled bit is '0' no input pin is connected. In case of a '1' the input is active. The first two bits in the first byte are reserved for the clock and the reset pin. The array OP is structured in the same manner as the array IP. An example for OP is illustrated in (5.4).

$$OP = \left[\underbrace{00000011}_{1.Byte}\ \underbrace{00000010}_{2.Byte} \ldots \underbrace{00000000}_{8.Byte} \right] \tag{5.4}$$

Here, three outputs are used which are situated in the first byte. If the polled bit is '0' no output pin is active. In case of a '1' the output is connected. Normally, IP and OP are results from the determination of pin types described in Section 3.2. For the purpose of analysis the arrays can be prepared by hand. After describing the input-output settings an example of a nonlinear FSM will be given in the next section to demonstrate the correct operation of the nonlinear analysis.

5.4.2 Example of a Nonlinear Non-Reduced Moore FSM

Before the nonlinear analysis can be started two additional conditions have to be considered. First, the analysis requires strongly connected FSMs [Moor56]. Otherwise, a complete analysis is not possible. However, an automaton is strongly connected if for any ordered pair of states (z_0, z_1) a sequence of inputs exists which will take the machine from state z_0 to z_1 [ibid]. Any strongly connected FSM has a state transition diagram which is a connected graph. In case of a not strongly connected graph it is possible that some states cannot be reached. Secondly, only the function of reduced automata will be obtained. In case of a non-reduced automata the nonlinear analysis automatically obtains the reduced function of the unknown IC under investigation. To demonstrate this function an example of a non-reduced FSM [ibid] was chosen which was implemented into the analysis environment. Table 5.7 shows the state transition table as well as the output function of the non-reduced example. Here, it can be seen that the model exists of one input, one output and three states. Furthermore, the automaton to be investigated is a Moore automaton.

	IW = 0	IW = 1	OW
z^n	z^{n+1}	z^{n+1}	
0	1	0	0
1	2	0	1
2	1	0	0

Table 5.7: State Transition Table and Output Function of the Non-Reduced Example

Now, the state transition table as well as the output function from Table 5.7 are taken and another description of the automaton is shown in Figure 5.15.

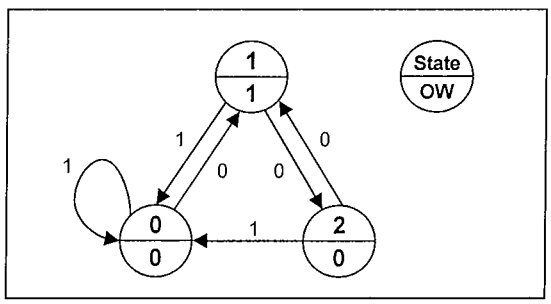

Figure 5.15: State Diagram of a Non-Reduced FSM [ibid]

Here, at a first glance it cannot be seen that the state diagram of the automaton presented in Figure 5.15 is non-reduced. Starting the nonlinear analysis the algorithm detects a Moore automaton. Furthermore, it gathers the following state transition table as well as output function as illustrated in Table 5.8.

	IW = 0	IW = 1	OW
z^n	z^{n+1}	z^{n+1}	
0	1	0	0
1	0	0	1

Table 5.8: State Transition Table and Output Function of the Example

To illustrate the solution of the algorithm in Figure 5.16 the state diagram can be seen which results from Table 5.8.

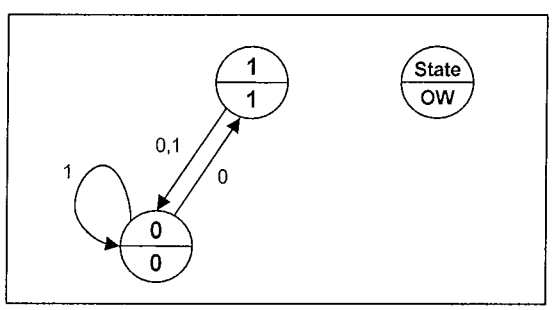

Figure 5.16: State Diagram of the Reduced FSM

Comparing Figure 5.15 and Figure 5.16 it can be seen, that state '2' can be omitted without any loss of the overall function. This example shows the reduction which is accomplished by the novel algorithm introduced in Section 4.5. Moreover, using the example described it was shown that not the structure but the function of a nonlinear unknown IC is determined. After having explained the reduction of the nonlinear algorithm an example of a nonlinear Moore FSM will be given in the next section.

5.4.3 Example of a Nonlinear Moore FSM

In this section a nonlinear IC model of type Moore is introduced. It consists of 16 states, one input and five outputs. The more outputs exist the more information are available to identify the unknown IC. Table 5.9 shows the state transition as well as the output function.

	IW = 0	**IW = 1**	**OW**
z^n	z^{n+1}	z^{n+1}	
0	0	1	0
1	2	3	1
2	4	5	2
3	6	7	19
4	8	9	4
5	10	11	5
6	12	13	6
7	14	15	23
8	0	1	8
9	2	3	9
10	4	5	10
11	6	7	27
12	8	9	12
13	10	11	13
14	12	13	14
15	14	15	31

Table 5.9: State Transition Table and Output Function of the Example

The state diagram of Table 5.9 is illustrated in Figure 5.17. Each binary output out_0, out_1, ..., out_4 can be summarised as a decimal output word, where out_4 relates to the most significant bit (MSB) and out_0 is the least significant bit (LSB). Obviously, the FSM is of type Moore because each state generates only one output word.

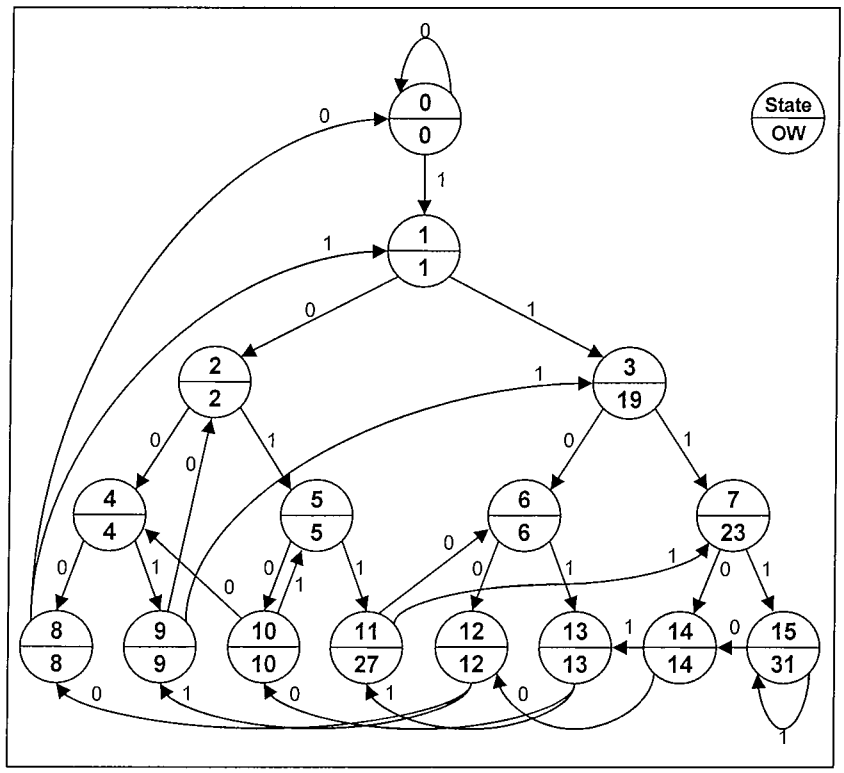

Figure 5.17: State Diagram of the Example

In the top right corner in Figure 5.17 state the current state and output word is the associated output word. After the reset the FSM is in state 0 with $OW = 0$. From state 0 all possible paths are passed through. The state diagram was implemented into hardware using the design software Quartus II. Here, it represents an unknown IC which had to be identified. After the implementation the unknown IC consists of five flip-flops and an AND gate which is illustrated in Figure 5.18. Additionally, there are a clock and a reset pin to ensure correct operation.

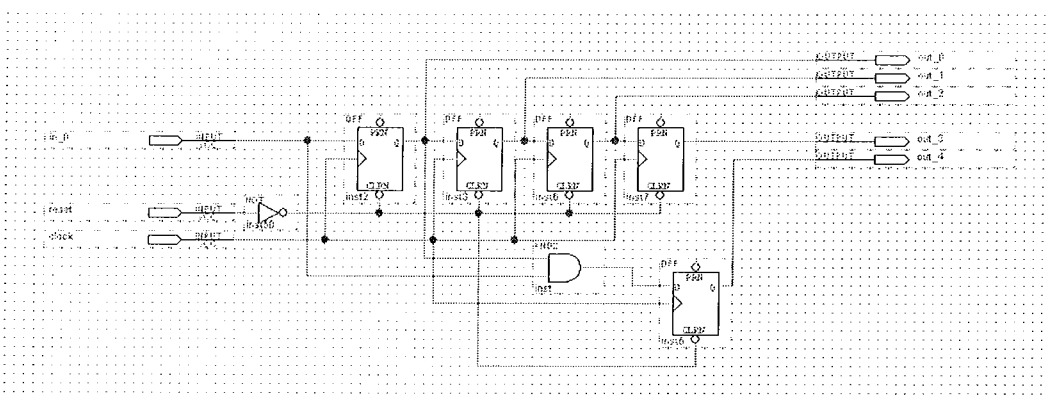

Figure 5.18: Nonlinear Moore FSM Implemented

Now, from Figure 5.18 the state transitions and the output function have to be identified. Furthermore, the type of automaton has to be determined. As previously described in Chapter 4 the analysis starts with the separation into a Moore or Mealy automaton. Here, a random input word is applied and then a clock pulse is caused. Afterwards, the first input word '0' is applied to the automaton and the resulting output word is recorded. In the following all other input words are applied to the circuit without a clock pulse. Next, the output words which appear are compared to the stored one. Using the example implemented all output words are equal. Therefore, it is classified as Moore automaton.

After the classification into the Moore automaton the preparing algorithm as described in Section 4.4 starts. Here, random input words are applied and a clock pulse is generated afterwards. Since, the example under investigation is a Moore automaton set {1} of $OW(t\text{-}1)$; IW; $OW(t)$ from (4.10) is taken. In constant intervals a reset is applied to restart from defined initial states. During the preparation the previous and the following output word are recorded at each step. Additionally, the input word which has caused the step is stored. However, the unknown IC is identified as a Moore FSM. Therefore, no output word list is generated because each state only produces one output word. Moreover, the algorithm uses the automatic approximation of states. Therefore, no information of the maximum number of states was given. After this preparation the main algorithm described in Section 4.5 has determined the state transition table as well as the output function. Table 5.10 shows the results of the nonlinear analysis.

	IW = 0	IW = 1	OW
z^n	z^{n+1}	z^{n+1}	
0	0	1	0
1	2	3	1
2	4	5	2
3	6	7	19
4	8	9	4
5	10	11	5
6	12	13	6
7	14	15	23
8	0	1	8
9	2	3	9
10	4	5	10
11	6	7	27
12	8	9	12
13	10	11	13
14	12	13	14
15	14	15	31

Table 5.10: State Transition Table and Output Function after Nonlinear Analysis

Table 5.9 and Table 5.10 are now checked for consistency. Due to conformance the nonlinear analysis procedure has resulted in correct behaviour of the IC investigated. Table 5.11 shows the results of the nonlinear example described using both hardware analysis as well as simulation.

	Hardware Analysis	Simulation
Detected Automaton	Moore	Moore
NoS	16	16
Evaluation Time	46173,0s (= 12,83h)	8,74s

Table 5.11: Result Table of Different Simulation Modes

Here, the analysis has resulted in the determination of the correct type of automaton. Furthermore, the algorithm found the correct number of states. Without any prior knowledge of the number of internal states the presented algorithm is also able to find the correct state transition table as well as the output function. After having explained a nonlinear Moore unknown FSM an example of a Mealy automaton will be described in the next section.

5.4.4 Example of a Nonlinear Mealy FSM

In this section an example of a nonlinear Mealy FSM will be investigated. Here, the ISCAS model S27 is used which consists of five states, four inputs G0 to G4 and the

output G17. In the example only one output is used. Therefore, the information content is not high because only the binary values '0' and '1' can occur. Table 5.12 shows the state transition table and the output word combination.

z^n	IW																OW
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	Nr.
	z^{n+1}																
0	0	1	2	2	0	1	0	1	3	3	2	2	3	3	0	1	2
1	0	1	2	2	0	1	0	1	0	1	2	2	0	1	0	1	3
2	2	2	2	2	0	1	0	1	2	2	2	2	0	1	0	1	3
3	3	1	4	2	3	1	3	1	3	3	4	2	3	3	3	1	0
4	4	2	4	2	3	1	3	1	4	2	4	2	3	1	3	1	1

Table 5.12: State Transition Table and Output Function of the Benchmark S27

The model S27 investigated is a Mealy automaton. Therefore, an output word combination (OWC) per state exists. These combinations are referenced by the output word list as illustrated in Table 5.13.

OW	IW															
Nr.	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	y^{n+1}															
0	0	1	0	1	0	1	0	1	0	0	0	1	0	0	0	1
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
2	1	1	1	1	1	1	1	1	0	0	1	1	0	0	1	1
3	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Table 5.13: Output Word Combination of Model S27

After synthesising, the model of the IC was implemented into the FGPA which represents the unknown IC. Additionally, a reset capability was added. This implementation of the model S27 is illustrated in Figure 5.19.

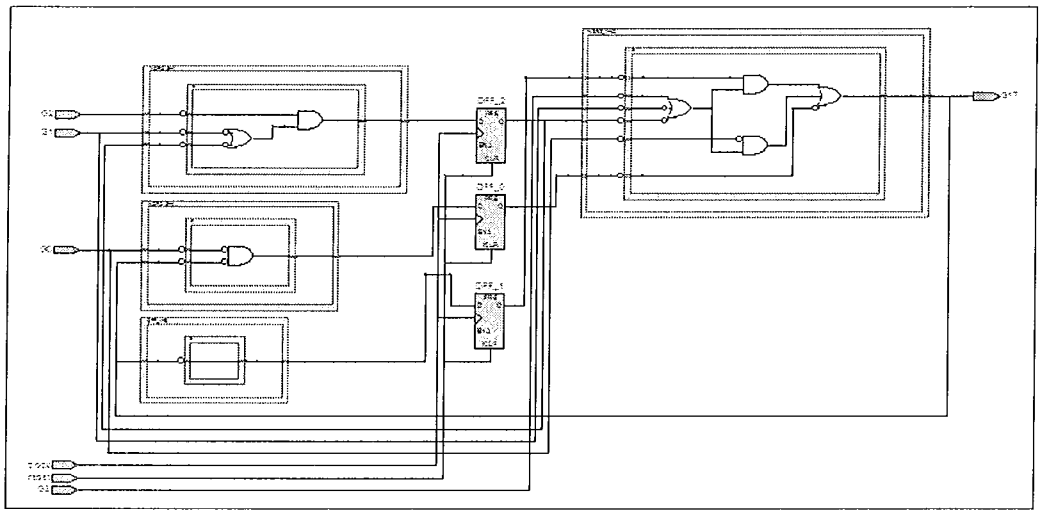

Figure 5.19: Model S27 Implemented

In this figure, it can be seen that the implementation S27 consists of three flip-flops as well as several logic gates.

The analysis starts with the separation into Moore or Mealy automaton. Here, a random input word is applied and then a clock pulse is caused. After this step, the first input word is applied to the automaton. Next, the resulting output word is recorded. Moreover, all other input words are applied to the circuit without a clock pulse applied. Because of the logic elements shown in Figure 5.19 variations of the output word occur. Therefore, the example S27 is classified as a Mealy automaton. After the classification into a Mealy automaton the preparation algorithm as described in Section 4.4 is started. Since, the example under investigation is a Mealy automaton set {2} of $OWC(t\text{-}1); IW$; the $OWC(t)$ from (4.10) is taken. The determination of the number of distinguishable states using the most significant input word has to be carried out in the next step. Example S27 produces four most significant input words which have a *significance* = [1, 2, 1, 1]. Taking Equation (4.11) and using MSIW and *significance* the number of distinguishable states can be computed as shown in Equation (5.5).

$$\begin{aligned} NoDS &= \sum_{MSIW} significance \\ &= 1+2+1+1 \\ &= 5 \end{aligned} \tag{5.5}$$

Here, the number of distinguishable states is equal to five. This number is equal to the real number of states. Therefore, the automaton S27 can be identified directly.

After the preparation the main algorithm determines the length of the tree as was shown in Equation (4.15). To determine this length the number of states has to be computed first using Equation (4.12). The number of states can be calculated as in Equation (5.6).

$$NoS = NoDS + 2 = 5 + 2 = 7 \tag{5.6}$$

Now, the length of the tree can be calculated as shown in (5.7).

$$length_{tree} = NoS - NoDS + 2 = 7 - 5 + 2 = 4 \tag{5.7}$$

As a result the length of one investigation tree is equal to four. After the determination of all parameters the fast identification starts as was illustrated in Figure 4.5. Using the example S27 five different state trees are generated and recorded. After each investigation cycle the number of found states is incremented by one. Furthermore, the path detected is recorded from its initial state to the new state. It is the information how the new state can be reached again. Moreover, the state is entered as new. If no new state is found the investigation of the state is finished. After executing the main algorithm once the algorithm categorises the result as valid as shown in part {1} of Equation (4.22) and no further investigation has to be carried out. Figure 5.20 shows the screen shot of MATLAB's analysis of the benchmark S27.

```
ans =
Testing the automaton for being Moore or Mealy...

ans =
Detected a Mealy-Automaton, gathering first information...

ans =
"nos" set from 0 to 7

ans =
Starting Tree-Algorithm...

ans =
nos: 7   nods: 5         current state: 1        detected states: 1

ans =
nos: 7   nods: 5         current state: 2        detected states: 4

ans =
nos: 7   nods: 5         current state: 3        detected states: 4

ans =
nos: 7   nods: 5         current state: 4        detected states: 4

ans =
nos: 7   nods: 5         current state: 5        detected states: 5

Elapsed time is 636.197648 seconds.
```

Figure 5.20: Analysis of Example S27

Several information can be taken from Figure 5.20. The user is always informed about the status of the nonlinear procedure. At the end five states were detected.

Furthermore, the evaluation time is printed on screen as well as the results are stored in a file. The novel procedure was able to determine the correct type of automaton. Moreover, the algorithm found the correct number of states. Without any knowledge of the number of internal states the proposed algorithm was also able to find the correct state transition table as well as the output function. After having shown several examples of unknown FSMs the memory requirements of the system will be discussed in the next section.

5.4.5 Memory Requirements

To describe the performance of the overall nonlinear analysis the memory requirements have to be also considered. However, it is difficult to determine the exact amount of memory required for the simulation because it cannot be excluded that other software applications and processes run in the background. Moreover, MATLAB itself requires significant memory resources independent of the analysis procedure. During the simulation several variables are responsible for the allocation of memory. Therefore, the memory used can be approximated. The tree array is the most memory consuming variable which will be considered now in detail. To demonstrate the calculation of the memory requirement using the fast identification the already introduced benchmark S27 was chosen. First, the number of input words has to be computed. Therefore, Equation (4.7) is used to calculate the number of input words as shown in (5.8).

$$NoIW = 2^{NoUIP} = 16 \tag{5.8}$$

Again, Equation (4.15) then is used to determine the length of the state tree where the number of distinguishable states is equal to five and the number of states is equal to seven. Therefore, the length of the tree is equal to four as shown in (5.9).

$$length_{tree} = NoS - NoDS + 2 = 7 - 5 + 2 = 4 \tag{5.9}$$

After the determination of the number of input words as well as the length of the tree these variables have to be input into Equation (4.17). The result can be seen in Equation (5.10).

$$length_{treearray} = \sum_{n=0}^{length_{tree}-1} NoIW^n = \sum_{n=0}^{3} 16^n = 4369 \tag{5.10}$$

Here, the length of one tree array is equal to 4396. During the simulation for each state one tree array exists. Therefore, using Equation (4.16) the number of trees can be computed as shown in (5.11)

$$NoTr\left(maximum\ length\right) = NoS + 1 = 5 + 1 = 6 \tag{5.11}$$

where the added one is a temporary tree which is deleted after the identification procedure. Using MATLAB each cell of the array requires a space of 8Byte. The length of the tree array is now multiplied by the number of trees as well as by the memory for one cell in an array as in Equation (5.12).

$$\begin{aligned} memory\left(tree\right) &= length_{treearray} \times memory\left(length_{treearray}\right) \times NoTr \\ &= 4369 \times 6 \times 8\text{Byte} \\ &= 209712\text{Byte} \end{aligned} \tag{5.12}$$

Furthermore, several variables exist which can be approximated as constants. In this case they require 24000Byte of user memory. Adding the memory of the trees and the remaining variables a memory of about 230MByte is required for the simulation. Moreover, MATLAB itself uses for its running about 150MByte. The standard WINDOWS applications requires about 150MByte. Therefore, about 530Mbyte of memory is required to perform the fast identification of the example S27. This value is well below 2GByte RAM which was used for simulation as described in Section 5.1. Therefore, no problems did occur during simulation. However, Equation (5.13) shows how the memory for simulation can be computed in reverse way.

$$\begin{aligned} memory_{simulation} &= 2048\text{MByte}\left(\text{PC RAM}\right) \\ &\quad -150\text{MByte}\left(\text{WINDOWS}\right) \\ &\quad -200\text{MByte}\left(\text{MATLAB incl. constants}\right) \\ memory_{simulation} &= 1698\text{MByte} \end{aligned} \tag{5.13}$$

Here, about 1700MByte might be allocated by the fast simulation. The nonlinear analysis always starts using the fast identification of nonlinear FSMs. In case of

limited user memory a modification of the fast identification exists which was described in Section 4.5.3. Therefore, it is possible to also analyse complex unknown ICs. Here, the nonlinear analysis procedure automatically switches from the fast to slow solution if the tree length is too long for a calculation using the adjusted memory. This value can be adapted in relation to the available user memory. Nowadays, standard PCs can manage 16GByte or more memory. Here, the use of more than 15GByte is possible to perform the fast analysis of nonlinear unknown ICs. Therefore, these systems are able to analyse even very complex ICs without any constraints. After having described the memory requirements with respect to the benchmark S27 the results of the simulation as well as the hardware analysis will be given in the next section.

5.4.6 Simulation and Hardware Analysis of IC Models

In this section the results of the nonlinear identification procedure will be discussed. Using the analysis environment which was described in Section 5.1.1 this identification procedure is verified using both simulation and hardware. The IC models were analysed with models having unknown as well as known number of internal states. The following tables will show the results of the simulation and the hardware analysis of the nonlinear identification procedure. Furthermore, for each model the result with unknown as well as known number of states is shown. Table 5.14 first shows the results where NoFS is the number of found states. Moreover, STT is the state transition table and OF represents the output function. Here, the number of internal states of the IC models to be investigated is unknown.

Circuit Name	Type of FSM	FSM Found	NoS	NoFS	STT Found?	OF Found?	Evaluation Time
EC1	Mealy	Mealy	1	1	yes	yes	7713,6s = 2,14h
ELS1	Mealy	Mealy	8	8	yes	yes	62110,0s = 17,25h
ENLS1	Mealy	Mealy	8	8	yes	yes	62127,0s = 17,26h
S27	Mealy	Mealy	5	5	yes	yes	615580,0s = 7,12d
B06	Moore	Moore	13	13	yes	yes	11456s = 3,18h

Table 5.14: Results of Hardware Analysis Using Unknown Number of States

As can be seen in Table 5.14 in each case the nonlinear algorithm found the correct type of unknown IC. The evaluation time in the right column shows that about one week is needed to identify the benchmark S27. The other IC models can be evaluated in less than a day. However, it is even possible to identify the expected state transition table as well as the correct output function.

Table 5.15 presents the simulation results using unknown number of internal states of the IC models to be investigated.

Circuit Name	Type of FSM	FSM Found	NoS	NoFS	STT Found?	OF Found?	Evaluation Time
EC1	Mealy	Mealy	1	1	yes	yes	13,6s
ELS1	Mealy	Mealy	8	8	yes	yes	73,4s
ENLS1	Mealy	Mealy	8	8	yes	yes	70,7s
S27	Mealy	Mealy	5	5	yes	yes	636,2s
B06	Moore	Moore	13	13	yes	yes	17,2s
C17	Mealy	Mealy	1	1	yes	yes	2378,7s = 39,6min

Table 5.15: Results of Simulation Using Unknown Number of States

Instead of hardware analysis the simulation of IC models was chosen. It can be seen from Table 5.15 that the algorithm found the correct type of all unknown ICs under investigation. Moreover, the correct number of states was always found. Hence, the correct state table as well as the correct output function were in all cases successfully found. However, the algorithm introduced was developed to analyse nonlinear FSM. As can be seen from Table 5.15 combinatorial as well as linear sequential FSM can

also be identified using the novel algorithm. Furthermore, the evaluation time in the right column shows that the simulation is accomplished within less than an hour for even complex circuits.

In case that the exact number of states is known both the hardware analysis shown Table 5.14 as well as the simulation as presented Table 5.15 was used. Therefore, the same implementations were analysed with known number of states instead the initial number of states equal to zero. First, Table 5.16 presents the hardware models analysed using the known number of states.

Circuit Name	Type of FSM	FSM Found	NoS	NoFS	STT Found?	OF Found?	Evaluation Time
EC1	Mealy	Mealy	1	1	yes	yes	282,7s = 4,7min
ELS1	Mealy	Mealy	8	8	yes	yes	39124,0s = 10,9h
ENLS1	Mealy	Mealy	8	8	yes	yes	39117,0s = 9,9h
S27	Mealy	Mealy	5	5	yes	yes	466562,0s = 5,4d
B06	Moore	Moore	13	13	yes	yes	2758,0s = 46,0min

Table 5.16: Results of Hardware Analysis Using Known Number of States

From Table 5.16 it can be seen that in each case the nonlinear detection algorithm found the correct type of the unknown IC. In comparison to the hardware analysis using an unknown number of states the identification using known number of states is 1,3 times faster. Here, no approximation of the number of states is necessary which was explained in Section 4.5. Furthermore, the analysis identified the expected state transition table as well as the output function.

Table 5.17 shows the simulation results using known number of states.

Circuit Name	Type of FSM	FSM Found	NoS	NoFS	STT Found?	OF Found?	Evaluation Time
EC1	Mealy	Mealy	1	1	yes	yes	0,355s
ELS1	Mealy	Mealy	8	8	yes	yes	34,3s
ENLS1	Mealy	Mealy	8	8	yes	yes	34,0s
S27	Mealy	Mealy	5	5	yes	yes	8,74s
B06	Moore	Moore	13	13	yes	yes	2,94s
C17	Mealy	Mealy	1	1	yes	yes	2048,4s = 34,1min

Table 5.17: Results of Simulation Using Known Number of States

From Table 5.17 it can be seen that the nonlinear algorithm firstly investigates the type of automaton. Afterwards, the state transition table as well as the output function of the unknown IC were analysed. From all result tables presented Table 5.17 exhibits the lowest evaluation times of the IC models analysed. This can be explained that in this case the number of states was known prior to simulation.

It can be concluded from the overall results that the simulation is faster than the hardware analysis. This can be explained with the transfer protocol between MATLAB and the development board. All vectors which have to be applied to the unknown IC have to be coded for transmission. Furthermore, MATLAB has to transfer the data through different instances in WINDOWS. However, in both cases the algorithm proposed is able to find the type, the state transition table as well as the output function of the investigated unknown IC within a fine time. To approximate the hardware analysis time from only the simulation results an equation was developed. Here, the average factor $k_{\varnothing}$ is used which is calculated in Equation (5.14).

$$k_{\varnothing} = \frac{\sum_{1}^{n} \frac{t_{n_{hardware}}}{t_{n_{simulation}}}}{n} \tag{5.14}$$

In this equation, each evaluation time of hardware analysis is divided by the evaluation time of simulation. Afterwards, all factors were added up to n where n is the number of ICs investigated. After applying all values of the results investigated to Equation (5.14) the average factor is calculated as in (5.15).

$$k_{\varnothing} = 6133 \tag{5.15}$$

From (5.15) it can be concluded that the evaluation time of the simulation is about 6133 times faster than the hardware analysis. To make the real hardware analysis as fast as possible the average factor $k_{\varnothing}$ has to be as small as possible. However, if only the simulation result is available $k_{\varnothing}$ has to be multiplied by the evaluation time. After having shown the results of the nonlinear analysis the timing behaviour of this procedure will be considered in the next section.

5.4.7 Timing Behaviour of the Nonlinear Analysis

As described in the previous section the timing behaviour is an important parameter of the overall IC analysis. Therefore, twelve Moore FSMs with two inputs and three outputs were implemented into hardware. Each automaton consists of a similar structure where the number of states was increased from one to twelve for each automaton. The results of this simulation are presented in Table 5.18. After the analysis the evaluation times were compared. First, all results are obtained where the algorithm knows the internal number of states. Secondly, all automata were analysed again without any knowledge about the number of internal states.

Circuit Name	NoS	Evaluation Time [s] (known NoS)	Evaluation Time [s] (unknown NoS)
M1	1	0,6024	26,0064
M2	2	0,8973	26,0835
M3	3	1,3394	26,2101
M4	4	1,7073	26,2979
M5	5	2,0950	26,4763
M6	6	2,4817	26,6030
M7	7	2,6592	26,8375
M8	8	2,5107	26,8166
M9	9	3,7475	26,9412
M10	10	9,1356	156,2405
M11	11	35,7305	677,1401
M12	12	159,4657	3118,7450

Table 5.18: Results of Calculation Time of the Simulation

In Figure 5.21 the results from Table 5.18 are illustrated.

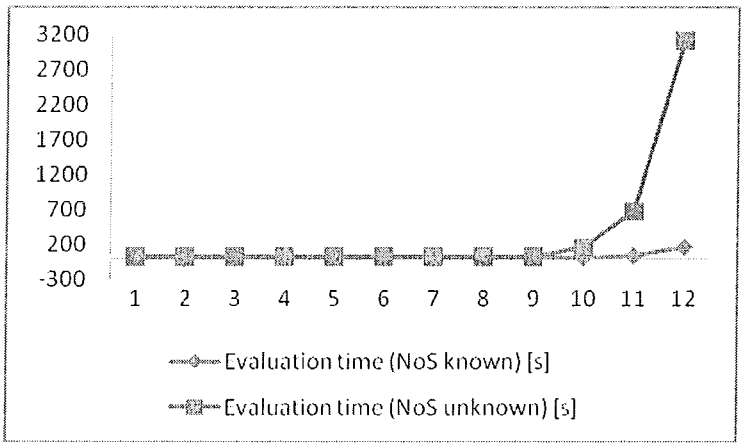

Figure 5.21: Timing Behaviour of the Nonlinear Analysis

From this figure, it can be concluded that if the number of states increases the evaluation time of automata with known number of states increases exponential. This can be explained with the increasing number of state trees to be investigated. Moreover, the algorithm switches from the fast to the slow analysis as was explained in Section 4.5.3. However, using the nonlinear algorithm with unknown number of states the timing behaviour increases also an exponential. This is because of the repeated evaluation of the algorithm to find a valid solution which was described in detail in Section 4.5. Nevertheless, an identification of the IC under investigation is always possible. In comparison traditional procedures require the prior knowledge of the number of internal states for a correct operation, which is for real unknown ICs an unrealistic assumption.

5.5 Summary

This chapter has presented the results of the separation procedure as well as for the nonlinear analysis of unknown ICs. To non-destructively simulate and analyse fully unknown ICs an analysis environment was introduced in the first section. Here, the user interface as well as the development board were described in detail which are connected to the unknown IC through an interface. Then, Section 5.2 presented the IEEE ISCAS benchmarks as well as user defined IC models which were used to validate the novel analysis procedures. In Section 5.3 the results of the separation procedure introduced in Chapter 3 were given. First, it was demonstrated that the proposed procedure is able to correctly detect all independent blocks within unknown ICs which is important to correctly identify any IC under test. Secondly,

the behaviour of each independent block found was investigated. To increase the complexity of the tests IEEE benchmark models were combined onto a single IC. Even with these very complex systems the overall separation procedure always detected the correct number of independent blocks. Furthermore, the presented algorithm always found the correct behaviour and logic operation of each IC under investigation.

The results of the novel nonlinear analysis introduced in Chapter 4 were given in Section 5.4. Again, the novel algorithm was applied to the benchmarks as well as user defined ICs. It was shown that the separation into Moore or Mealy was successfully carried out and that this significantly shorted the subsequent nonlinear analysis. Then it was shown that all functions included in the IC models under test were correctly identified. Moreover, the state transition tables as well as the output functions were successfully found. In this context, also the timing required by the presented system as well as its memory requirements were investigated. Here it was shown that the novel algorithm presented is always able to determine the functionality of a circuit within a finite time. Finally, it can be said that the nonlinear analysis procedure developed is able to fully identify all models of nonlinear FSMs. After discussing the results the next chapter will present the conclusions of this work.

6 Conclusions

After having presented the verification, tests and benchmarking of the analysis procedures presented to identify fully unknown CMOS ICs this chapter now summarises and analyses the overall work presented in this thesis. The first section presents the specific conclusions that can be directly taken drawn from the research. Then Section 6.2 builds on this research specific conclusions and transfers the outcomes of this work to other fields of interest. Finally, Section 6.3 then concludes with several possibilities of how to develop this research further and extend it to future projects.

6.1 Specific Conclusion

This thesis has presented a novel non-invasive approach to fully analyse unknown digital CMOS integrated circuits. Traditional methods proposed so far have had several disadvantages. Destructive procedures for example can only be used once to investigate an IC. Therefore, when using these destructive methods errors must not occur during the investigation. Otherwise, the information of the IC under investigation is irrecoverably lost. On the other hand traditional non-invasive procedures did not provide a complete analysis of a fully unclassified IC. Moreover, traditional methods only describe how to determine certain aspects of the ICs functionality and often also require some prior knowledge of the system. First, in this research the novel determination of the most common pin types of CMOS ICs was developed to identify the function of the unknown IC only from its input-output behaviour. This determination is based on the electrostatic discharge phenomenon. Originally, the electrostatic discharge protection was developed to protect the internal circuits of CMOS ICs against unexpected damages. In this research the electrostatic discharge protection circuits were for the first time used to determine the most common pin types as well as their location of CMOS ICs, thus, overcoming the first problem when investigating fully unclassified ICs.

After the determination of pin types was successfully solved a preliminary classification was introduced to investigate all possible structures as efficient as

mathematically possible. Generally, almost all unknown ICs consist of several independent internal circuits which in addition often exhibit a different behaviour. This complexity is difficult to handle for traditional analysis procedure. Furthermore, it is important to correctly analyse each independent block. Therefore, a separation procedure was developed and introduced in Section 3.3 which partitions the overall analysis. First, optimised test vector were developed as was shown in Section 3.3.1. The use of the so called binary maximum sequences were the key to the separation of the independent circuits. These sequences were generated and applied to the inputs of the unknown ICs. The novelty of these sequences was that each sequence has a uniform distribution of values '0' and '1' and causes a high switching activity inside the unknown IC. Afterwards, the outputs of the IC under test were observed and evaluated. The results have clearly shown that for the first time an accurate identification of the number of independent blocks in an IC is possible. Moreover, this technique has the advantage that the evaluation time can be significantly reduced. After the separation of the unknown IC into independent blocks had been presented a classification algorithm determining the ICs behaviours was introduced in Section 3.3.6. This new classification procedure is able to distinguish between combinatorial and sequential logic for each independent block. After that this classification algorithm is furthermore, able to determine linear sequential and nonlinear sequential behaviour. Again, the introduced binary maximum sequences were applied to the inputs of the unknown IC and the responses at the outputs were evaluated. It was shown that the novel separation procedure is able to successfully determine combinatorial, sequential linear or nonlinear behaviour of any unknown system. After determining the independent blocks as well as the general behaviour the most suitable identification procedure for each system could now be applied.

A large number of unknown ICs have a nonlinear behaviour. Therefore, this thesis has proposed a novel identification procedure to fully determine nonlinear ICs which consist of several improvements compared to traditional procedures. Traditional identification procedures always required some prior knowledge of the number of internal states to correctly determine the internal function of the IC under investigation. In practice the number of states is however, not always known. Therefore, a novel iteration procedure was developed by which the algorithm

independently approximates the number of states. Furthermore, an automatic separation into Moore or Mealy automaton was developed, which is based on their different logic structures. This separation was achieved by applying random input words with and without a clock to the unknown system. From the output responses of the automata their behaviour was then determined. Moreover, a modification of the identification procedure was introduced which is also able to identify the IC under test even if the memory required to identify very large systems runs low. Here, a slow identification procedure was introduced which significantly reduces the storage requirements of the algorithm. Therefore, the nonlinear analysis procedure can also be used to identify very complex ICs. Today most ICs have a reset capability. However, for not resettable ICs an approach was developed to determine an entry point while using the same procedure as for resettable systems. Furthermore, it was shown in Section 4.5.2 that the novel nonlinear identification procedure detects and eliminates redundant states. Therefore, the algorithm returns an optimised description of the IC under test.

To non-destructively analyse real unknown ICs only from its input-output behaviour an analysis environment was developed and built. Here, the user interface as well as the hardware developed were presented to analyse real unknown ICs. The algorithms introduced in Chapter 3 and Chapter 4 were then implemented into this analysis environment to demonstrate their correct operation. Furthermore, IEEE ISCAS benchmarks as well as user defined IC models were introduced to demonstrate the correct operation of the novel algorithm proposed in this thesis. The results of the tests have clearly shown that the proposed separation procedure is able to detect all independent blocks in any unknown IC without any problem. Furthermore, the separation procedure returned the correct behaviour of all ICs investigated. It was shown that the novel separation procedure is furthermore, able to successfully determine combinatorial, sequential linear and nonlinear behaviour of any unknown IC. In addition, the models of unknown ICs were combined to increase the complexity. Even with these very complex systems the classification procedure always identified the correct number of independent blocks within less than one hour. Moreover, the novel algorithm identified the correct behaviour of each IC under test. Then the results of the nonlinear analysis were presented. Here, the novel

algorithm was applied to the IEEE benchmarks as well as the user defined ICs. Again, the separation into Moore or Mealy was successfully carried out and therefore, the subsequent nonlinear analysis was significantly shortened. Secondly, all functions of the IC models under test were fully identified. Moreover, all state transition tables as well as all output functions were successfully found. Finally, the memory requirements of the presented algorithm were investigated using the IEEE benchmark S27. Here it was shown that even complex circuits can be investigated using a standard PC without any problem. Furthermore, an automatic detection of memory overruns was included into the algorithm. It was shown that in such a case the algorithm switches from fast to slow analysis, allowing even very complex unknown ICs to be fully determined. Therefore, it can be concluded that the novel algorithm presented in this thesis is able to successfully identify the nonlinear FSMs analysed. After discussing the specific conclusions the next section will explain the general conclusions of this research.

6.2 General Conclusions

In this research novel methods were developed to non-invasively analyse unknown digital CMOS ICs from its input-output behaviour only. After the determination of the pin types a classification procedure was introduced which separates the unknown IC at the earliest possible stage into different independent blocks and their respective behaviour. Furthermore, a procedure to identify nonlinear sequential unknown ICs was introduced. The proposed separation procedure as well as the nonlinear analysis is described by an abstract model of deterministic FSMs. Due to the abstract nature of the developed models, all procedures introduced in this thesis can easily be transferred to analyse any other unknown IC regardless of the type of material used. Therefore, it is possible to analyse other types of ICs which are built using techniques such as Bipolar, GaAs or BiCMOS.

Moreover, the application of the presented separation procedure is not just limited to the determination of unknown integrated circuits as was presented in this thesis. In general, it is possible to use the same algorithms to classify all applications based on the model of automaton. Here, theoretical as well as mathematical descriptions can be used to analyse these unknown models. Also nonlinear

sequential FSMs can be determined using the methods described in this thesis. Here again, the nonlinear analysis can be easily transferred to other applications. The procedure described in this work is for example also well suited to describe any control automata. Finite state machines are furthermore, used to model systems in diverse areas such as lexical analysis, pattern matching or communication protocols. In the example of communications protocols specification standard conformity checking is one of the main problems in protocol verification. Here, the presented algorithm can help to identify the different types of protocols and their implementation. It is furthermore, possible to directly use the nonlinear analysis procedure for software analysis. In this case, the software implemented could be optimised without any loss of information. In traditional software to program FPGAs it is not possible to detect FSMs in a reverse method. This however, would be useful to check the correct operation of an implementation. In this case, the nonlinear analysis procedure presented could be implemented to overcome this problem. After having described the general conclusion the future work will be presented in the next section.

6.3 Future Work

This thesis has presented the full classification of an unknown IC. However, there are some possible avenues for future work that may be considered. One research field could be a further optimisation of the nonlinear analysis procedure introduced. Here, decreasing the investigation depth of this identification procedure directly results in reducing the calculation time as well as a minimisation in memory requirements. This could be achieved by changing the examination of the results at the end of each investigation cycle. Using the nonlinear identification procedure introduced in Chapter 4 it was shown that the actual tree length is always computed using the number of distinguishable states plus two. Furthermore, the change of the number of states is the only variable in this computation. The alternative procedure could now include the number of states found as well as the previous tree length as shown in Equation (6.1).

$$length_{tree}(t) = NoS(t) - NoFS(t-1) + length_{tree}(t-1) \tag{6.1}$$

Here, the new tree length after each investigation cycle would be calculated as follows. The newly investigated number of states is decreased by the number of found states of the last investigation cycle. Moreover, the previous tree length is added. However, after the last investigation cycle more states could be distinguished in contrast only using a fixed number of states. The determination of the number of distinguishable states requires only a tree length of two. For a possible modified discrimination the value two is not sufficient. Therefore, for the correct determination the old value has to be replaced with a tree length which is equal to the new minimum investigation depth. It can be seen that this method decreases the tree length permanently. Using automata with a large and unknown number of states even a small difference in tree length can significantly reduce the calculation time as well as the system resources required. However, the expected number of investigated cycles would be increased. Therefore, a deeper investigation would be necessary.

A second possible improvement could be the implementation of the overall analysis procedures into hardware, for example an FPGA. Here, the implementation of the optimised procedures as a standalone application is conceivable. One part of the developed analysis environment is already implemented into an FPGA. Such an standalone FPGA system would achieve a maximum in evaluation speed. The main advantage would be that the test vectors required for the analysis of the unknown IC would be generated in real-time on the chip rather than be generated on a PC and transferred over a bus to the IC under test.

Another possible direction for further research could be a deeper investigation of other IC materials as well as electrostatic discharge structures. This could be interesting if the determination of pin types and their locations should be applied to ICs which are manufactured for example in Gallium Arsenide or a Bipolar technology. However, nowadays the electrostatic discharge structures are improved with each technology change. Therefore, a further investigation of this aspect might become necessary.

References

[Aho91] A. V. Aho, A. T. Dahbura, D. Lee, M. U. Uyar, "An optimization technique for protocol conformance test generation based on UIO sequences and rural Chinese postman tours," *IEEE Transactions on Communications*, Vol. 39, pp. 1604-1615, Nov. 1991.

[Aise67] M. A. Aiserman, L. A. Gussew, L. I. Rosonoer, I. M. Smirnowa, A. A. Tal, *Logik, Automaten, Algorithmen*, Berlin: Akademie-Verlag, 1967.

[Alma89] A. E. A. Almaini, *Kombinatorische und sequentielle Schaltsysteme*, Weinheim: VCH Verlagsgesellschaft mbH, 1989.

[Alsa06] U. Alsaiari, R. Saleh, "Testable and self-repairable structured logic design," in *Proceedings of the IEEE International Symposium on Circuits and Systems ISCAS 2006*, pp. 5623-5626, May 2006.

[Alte09] ALTERA Corporation [Online]. Available: http://www.altera.com. [Accessed: Feb. 2, 2009].

[Amer02] E. A. Amerasekera, C. Duvvury, *ESD in Silicon Integrated Circuits*, 2nd Edition, Hoboken, NJ: Wiley & Sons, 2002.

[Aste03] A. Asteroth, C. Baier, *Theoretische Informatik*, München: Pearson Studium, 2003.

[Atme09] ATMEL Corporation [Online]. Available: http://www.atmel.com/products/ULC/, [Accessed: Feb. 2, 2009].

[Auer96] A. Auer, R. Kimmelmann, *Schaltungstest mit Boundary Scan*, Heidelberg: Hüthig Verlag, 1996.

[Balk93] L. Balke, K. H. Böhling, *Einführung in die Automatentheorie und die Theorie Formaler Sprachen*, Mannheim: Wissenschaftsverlag, 1993.

[Bar92] U. Bar, J. M. Schneider, "Automated validation of TTCN test suites," in *Proceedings IFIP WG6.1 12th International Symposium on Protocol Specification, Testing and Verification*, Amsterdam, North-Holland, pp. 79-296, 1992.

[Bard50] J. Bardeen, W. H. Brattain, "Three-electrode circuit element utilizing semiconductive materials," U.S. Patent 2,524,035, Oct. 1950.

[Blyt93] S. Blythe, B. Fraboni, S. Lall, H. Ahmed, U. de Riu, "Layout reconstruction of complex silicon chips," *IEEE Journal of Solid-State Circuits*, Vol. 28, Issue 2, pp. 138-145, Feb. 1993.

[Boch75] D. Bochmann, *Einführung in die strukturelle Automatentheorie*, Berlin: VEB Verlag Technik, 1975.

[Boch81] D. Bochmann, C. Posthoff, *Binäre Dynamische Systeme*, Berlin: Akademie-Verlag, 1981.

[Bool54] G. Boole, *The Laws of Thought*, in *George Boole's Logical Works*, Vol. 2, Chicago, IL: Open Court Publishing, 1911, first published in 1854.

[Bour02] N. G. Bourbakis, A. Mogzadeh, S. Mertoguno, C. Koutsougeras, "A knowledge-based expert system for automatic visual VLSI reverse-engineering: VLSI Layout Version," *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 32, no. 3, pp. 428-436, May 2002.

[Bund05] B. Bundschuh, G. Werner, St. Schiemann, M. Werner, „Der fault-locator – Ein preisgünstiges fehlerortungsgerät auf korrelationsbasis für faseroptische verbindungen," in *Research Report 2004/2005 University of Applied Sciences Merseburg*, pp. 52-55, Oct. 2005.

[Char03] D. Chardonnereau, "BIST, BISR tools to push up quality, yield," in *EETimes* [Online]. Available:

http://www.eetimes.com/story/OEG20030428S0088, [Accessed: Feb. 2, 2009].

[Chik90] E. J. Chikofsky, J. H. Cross II, "Reverse engineering and design recovery: a taxonomy," *Software IEEE*, Vol. 7, Issue 1, pp. 13-17, Jan. 1990.

[Chow78] T. S. Chow, "Testing software design modelled by finite state machines," *IEEE Transactions on Software Engineering*, Vol. SE-4, Issue 3, pp. 178-187, May 1978.

[Corno00] F. Corno, M. S. Recorda, G. Squillero, "RT-Level ITC'99 benchmarks and first ATPG results," *IEEE Design and Test of Computers*, Vol. 17, Issue 3, pp. 44-53, July-Sept. 2000.

[Dabr98] S. Dabral, T. Maloney, *Basic ESD and I/O Design*, Malden, MA: Wiley-Interscience, 1998.

[Dewe97] A. M. Dewey, *Analysis and Design of Digital Systems with VHDL*, Boston: PWS Publishing Company, 1997.

[Fing97] A. Finger, *Pseudorandom-Signalverarbeitung*, Stuttgart: Teubner Verlag, 1997.

[Frie71] A. D. Friedman, P. R. Menon, *Fault Detection in Digital Circuits*, Englewood Cliffs, NJ: Prentice Hall, Dec. 1971.

[Gill66] A. Gill, *Linear Sequential Circuits*, New York, NY: McGraw-Hill Book Company, 1966.

[Gone80] G. Gonenc, "A method for the design of fault detection experiments," *IEEE Transactions on Computers*, Vol. C-19, pp. 551-558, 1980.

[Göss72a] M. Gössel, *Angewandte Automatentheorie I*, Berlin: Akademie Verlag, 1972.

[Göss72b] M. Gössel, *Angewandte Automatentheorie II*, Berlin: Akademie Verlag, 1972.

[Grea92] W. D. Greason, K. W. K. Chum, "Experimental determination of ESD latent phenomena in CMOS integrated circuits," *IEEE Transactions on Industry Applications*, Vol. 28, no. 4, pp. 755-760, July / Aug. 1992.

[Hans99] M. C. Hansen, H. Yalcin, J. P. Hayes, "Unveiling the ISCAS-85 benchmarks: a case study in reverse engineering," *IEEE Design and Test of Computers*, Vol. 16, Issue 3, pp. 72-80, July-Sept. 1999.

[Hans09] M. C. Hansen, H. Yalcin, J. P. Hayes, "ISCAS High-Level Models," [Online]. Available: http://www.eecs.umich.edu/~jhayes/iscas/, [Accessed: Feb. 9, 2009].

[Heis94] J. Heistermann, *Genetische Algorithmen: Theorie und Praxis evolutionärer Optimierung*, Stuttgart: Teubner Verlag, 1994.

[Henn64] F. C. Hennie, "Fault detection experiments for sequential circuits," in *Proceedings of the 5th Annual Symposium Switching Circuit Theory and Logical Design*, pp. 95-110, 1994.

[Holz90] G. J. Holzmann, *Design and Validation of Protocols*, Englewood Cliffs, NJ: Prentice-Hall, 1990.

[Hopc02] J. E. Hopcroft, R. Motwani, J. D. Ullman, *Einführung in die Automatentheorie, Formale Sprachen und Komplexitätstheorie*, 2nd revised Edition, München: Pearson Studium, 2000.

[Hunt04] E. V. Huntington, "Sets of independent postulates for the algebra of logic," *Transactions on American Mathematical Society*, Vol. 5, pp. 288-309, 1904.

[Ixen09] IXENTO [Online]. Available: http://www.ixento.com/, [Accessed: Feb. 2, 2009].

[Isca08] ISCAS Benchmark Circuits [Online]. Available: http://www.fm.vslib.cz/~kes/asic/iscas, [Accessed: Mar. 4, 2008].

[Jarz95] St. Jarzabek, T.P. Keam, "Design of a generic reverse engineering assistant tool," in *Proceedings of the 2nd Working Conference on Reverse Engineering*, Toronto, Canada, pp. 61-70, July 14-16, 1995.

[JTAG09] JTAG Technologies [Online]. Available: http://www.jtag.com/en/Learn/Standards/IEEE_1149.1, [Accessed: Feb. 9, 2009].

[Kats91] K. Katsuma, F. Sato, T. Nakakawaji, T. Mizuno, "Strategic testing environment with formal description techniques," *IEEE Transactions on Computers*, Vol. 40, pp. 61-70, July 514-525, Apr. 1991.

[Ker05] M.-D. Ker, K.-C. Hsu, "Overview of on-chip electrostatic discharge protection design with SCR-based devices in CMOS integrated circuits," *IEEE Transactions on Device and Materials Reliability*, Vol. 5, no. 2, pp. 235-249, June 2005.

[Kien06] U. Kiencke, *Ereignisdiskrete Systeme*, 2nd Edition, München: Oldenbourg Verlag, 2006.

[Kilb76] J. S. Kilby, "Invention of the integrated circuit," *IEEE Transactions on Electron Devices*, no. 7, pp. 648-654, 1976.

[Kim05] Y. C. Kim, V. D. Agrawal, K. K. Saluja, "Combinational Automatic Test Pattern Generation for Acyclic Sequential Circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 24, Issue 6, pp. 948-956, June 2005.

[Klin04] W. Klingenstein, "Semiconductor fundamentals and historical overview," in *Semiconductors*, 2nd revised Edition, Erlangen: Publicis Corporate Publishing, 2004.

[Knap08] K. H. Knapp, "Heute erforscht, morgen produziert," in *Elektronikreport - Forschung, Entwicklung, Konstruktion*, pp. 6-9, 2008 [Online]. Available:

http://www.cnt.fraunhofer.de/fhg/Images/ER0508FhGCNT_tcm208-125633.pdf, [Accessed: Feb. 9, 2009].

[Koha78] Z. Kohavi, *Switching and Finite Automata Theory*, 2nd Edition, New York, NY: McGraw-Hill Book Company, 1978.

[Kuro04] Y. Kuroe, "Learning and identifying finite state automata with recurrent high-order neural networks," *SICE Annual Conference*, pp. 2241-2246, Sapporo, Aug. 2004.

[Lack97] R. Lackes, *Neuronale Netze: Prinzipien und Anwendungen*, Bonn: Addison-Wesley Verlag, 1997.

[Laws04] M. V. Lawson, *Finite Automata*, Boca Raton, FL: Chapman & Hall / CRC, 2004.

[Lee93] D. Lee, K. Sabnani, "Reverse Engineering of communication protocols," in *Proceedings of the International Conference on Network Protocols*, pp. 208-216, Oct. 1993.

[Lee96] D. Lee, M. Yannakakis, "Principles and methods of testing finite state machines – a survey," *Proceedings of the IEEE*, Vol. 84, no. 8, pp. 1090-1123, Aug. 1996.

[Ligh58] M. J. Lighthill, *Introduction to Fourier analysis and generalised functions*, Cambridge: Cambridge University Press, 1958.

[Lipp05] H. M. Lipp, J. Becker, *Grundlagen der Digitaltechnik*, 5th Edition, München: Oldenbourg Verlag, 2005.

[Lu05] S.-K. Lu, J.-S. Shih, S.-C. Huang, „Design-for-testability and fault-tolerant techniques for FFT processors," *IEEE Transactions on Very Large Scale Integration Systems*, Vol. 13, no. 6, pp. 732-741, June 2005.

[Lunz06] J. Lunze, *Ereignisdiskrete Systeme*, München: Oldenbourg Verlag, 2006.

[Matl08] The MathWorks Incorporation [Online]. Available: http://www.mathworks.com, [Accessed: Dec. 11, 2008].

[Meal55] G. H. Mealy, "A method for synthesizing sequential circuits," in *Bell System Technical Journal*, Vol. 34, pp. 1045-1079, Sept. 1955.

[Mill91] R. K. Miller, S. Paul, "Generating minimal test length sequences for conformance testing of communication protocols," in *Proceedings of the 10th Annual Joint Conference of the IEEE Computer and Communications Societies Networking in the 90s*, Vol. 2, pp. 970-979, Apr. 1991.

[Moor56] E. F. Moore, "Gedanken-Experiments on Sequential Machines," in *Automata Studies*, Study 34, Princeton: Princeton University Press, 1956.

[Moor65] G. E. Moore, "Cramming more components onto integrated circuits" *Electronics,* Vol. 38, no. 8, pp. 114-117, Apr. 1965.

[Moor06] G. E. Moore, "Moore's Law at Forty," in *Understanding Moore's Law*, D. C. Brock, Editor, pp. 67-84, Philadelphia, PA: Chemical Heritage Foundation, 2006.

[Muel08a] W. Mueller, "Benchmarks for VHDL Simulation," [Online]. Available: http://jerry.c-lab.de/~wolfgang/VHDL/models/ISCAS/00.README.first., [Accessed: Feb. 9, 2009].

[Muel08b] W. Mueller, "Benchmarks for VHDL Simulation," [Online]. Available: http://jerry.c-lab.de/~wolfgang/VHDL/models/ISCAS/ISCAS89/, [Accessed: Feb. 9, 2009].

[Musl03] D. Musliner, M. J. Pelican, R. P. Goldman, "Incremental automata verification," International Patent WO 03/038690 A2, May 2003.

[Naka06] Y. Nakamura, J. Savir, H. Fujiwara, "BIST pretest of ICs: risks and benefits," in *Proceedings of the 24th IEEE Very Large Scale Integration Test Symposium*, pp. 144-149, Berkeley, CA, United States, Apr. / May 2006,.

[Nise09] Nisene Technology Group, [Online]. Available: http://www.nisene.com/ic_delayering.shtml, [Accessed: Feb. 9, 2009].

[Oppe04] A. V. Oppenheim, R. W. Schafer, J. R. Buck, *Zeitdiskrete Signalverarbeitung*, 2nd revised Edition, München: Pearson Studium, 2004.

[Patt06] R. S. Patty, "Three-dimensional integrated circuits and the future of system-on-chip designs," in *Proceedings of the IEEE*, Vol. 94, no. 6, June 2006.

[Pete67] W. W. Peterson, *Prüfbare und korrigierbare Codes*, München: Oldenbourg Wissenschaftsverlag GmbH, 1967.

[Phil09] Philips Semiconductors, Product Specification Integrated Circuit 74 HCT 00 [Online]. Available: http://pdf1.alldatasheet.com/datasheet-pdf/view/15521/PHILIPS/74HCT00.html, [Accessed: Feb. 9, 2009].

[Raza08] B. Razavi, *Fundamentals of microelectronics*, Hoboken, NJ: John Wiley & Sons, 2008.

[Reic01] J. Reichardt, B. Schwarz, *VHDL-Synthese*, 2nd revised Edition, München: Oldenbourg Wissenschaftsverlag GmbH, 2001.

[Reus69] B. Reusch, *Lineare Automaten*, Mannheim: Bibliographisches Institut Mannheim, 1969.

[Rich99] H. Richter, B. Bundschuh, S. Weh, "PN-CW Lidar using hardware correlator," in *Proceedings of IEEE International Geoscience and*

Remote Sensing Symposium, Hamburg, Germany, Vol. 1, pp. 491-493, June / July 1999.

[Schä08] S. Schäfer, "Schutz für empfindliche Chips," in *Markt & Technik - Die unabhängige Wochenzeitschrift für Elektronik*, Vol. 11, pp. 32-33, March 2008.

[Schl02] M. S. Schlansker, V. K. Kathall, G. Snider, S. A. Gupta, S. A. Mahlke, S. Abraham, "Automated design of processor using feedback from internal measurements of candidate systems" U.S. Patent 6,408,428 B1, June 2002.

[Schu67] D. Schulte, *Kombinatorische und Sequentielle Netzwerke / Grundlagen und Anwendungen der Automatentheorie*, München: Oldenbourg Verlag, 1967.

[Sedr04] A. S. Sedra, K. C. Smith, *Microelectronic Circuits*, Oxford: Oxford University Press, 2004.

[Shep63] J. C. Shepherdson, H. E. Sturgis, "Computability of recursive functions," in *Journal of the Association Computing Machinery*, Vol. 10, no. 2, pp.217–255, New York, NY, 1963.

[Shoc51] W. Shockley, "Circuit element utilizing semiconductive material," U.S. Patent 2,569,347, September 1951.

[Stür71] H. Stürz, W. Cimander, *Automaten, Theorie und Anwendung in der digitalen Schaltungstechnik*, Berlin: Verlag Technik, 1971.

[Tekm09] Tekmos [Online]. Available: http://www.tekmos.com/ASIC/FPGA-Conversions, [Accessed: Feb. 9, 2009].

[Tera09] Terasic Technologies [Online]. Available: http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=39&No=226, [Accessed: Feb. 9, 2009].

[Trak73] B. A. Trakhtenbrot, Ya. M. Bardzin, *Fundamentals in Computer Science Volume 1, Finite Automata – Behaviour and Synthesis*, North-Holland: North-Holland Publishing Company, 1973.

[Vasi73] M. P. Vasilevskii, "Failure diagnosis of automata," *Kibernetika*, no. 4, pp. 98-108, 1973.

[Wang06] X.-X. Wang, L.P. Liang, X.-J. Wang, "A new solution to implement multi-full-scan-chain test with JTAG," *8th International Conference on Solid-State and Integrated Technology*, Shanghai, China, pp. 2155-2157, Oct. 2006.

[Wuns86] G. Wunsch, *Handbuch der Systemtheorie*, Berlin: Akademie-Verlag, 1986.

[Wuns93] G. Wunsch, H. Schreiber, *Digitale Systeme*, 4th Edition, Berlin: Springer Verlag, 1993.

[Zema65] A. H. Zemanian, *Distribution Theory and Transform Analysis*, New York: McGraw-Hill, 1965.

Authors Publications

Patents

M. Brutscheck, B. Schmidt, A. Th. Schwarzbacher, St. Becker, "Verfahren zur identifizierung eines deterministischen, sequenziellen Automaten," Patent Application DE 10 2008 055 193.7, Dec. 2008.

Peer Reviewed Publications

[1] A. Th. Schwarzbacher, M. Brutscheck, O. Schwingel, J. B. Foley, "Constant divider structures of the form $2^n \pm 1$ for VLSI implementation," *Irish Signals and Systems Conference,* Dublin, Ireland, pp. 368-375, June 2000.

[2] A. Th. Schwarzbacher, J. B. Foley, M. Brutscheck, O. Schwingel, "Optimisation of real-time signal processing algorithms for low-power CMOS implementations," *International Symposium on Communication Systems, Networks and Digital Signal Processing 2000*, Bournemouth, United Kingdom, pp. 38-42, July 2000.

[3] M. Brutscheck, A. Th. Schwarzbacher, St. Becker, "Systematic analysis of unknown integrated circuits - ein kooperatives promotionsverfahren," *Research Report 2004/2005 University of Applied Sciences Merseburg*, p. 38, Oct. 2005.

[4] M. Brutscheck, St. Becker, M. Schwanecke, E. Rosenfeld, A. Kopp, "Verfahren zur elektronischen fokussierung von ultraschall unter verwendung von FPGA," *Research Report 2004/2005 University of Applied Sciences Merseburg*, pp. 39-41, Oct. 2005.

[5] M. Brutscheck, A. Th. Schwarzbacher, St. Becker, "A test environment for analysing unknown integrated circuits," *7th New Generation Scientist Conference*, Wernigerode, Germany, pp. 223-224, Jan. 2006.

[6] M. Schwanecke, E. Rosenfeld, A. Kopp, M. Brutscheck, "Laseroptische charakterisierung von fokussierten ultraschallfeldern," *7th New Generation Scientist Conference*, Wernigerode, Germany, pp. 77-80, Jan. 2006.

[7] M. Brutscheck, St. Becker, M. Schwanecke, E. Rosenfeld, E. Schölzel, A. Kopp, "Ein verfahren zur elektronischen fokussierung von ultraschall unter verwendung von programmierbaren logikbausteinen," *7th New Generation Scientist Conference*, Wernigerode, Germany, pp. 227-228, Jan. 2006.

[8] M. Brutscheck, M. Franke, A. Th. Schwarzbacher, St. Becker, "Determination of pin types and minimisation of test vectors in unknown CMOS integrated circuits," *13th Electronic Devices and Systems IMAPS CS International Conference*, Brno, Czech Republic, pp. 64-69, Sept. 2006.

[9] M. Brutscheck, M. Franke, A. Th. Schwarzbacher, St. Becker, "Determination of pin types and minimisation of test vectors in unknown CMOS integrated circuits," *Research Day 2006*, Merseburg, Germany, Nov. 2006.

[10]M. Franke, M. Brutscheck, A. Th. Schwarzbacher, St. Becker, "Analysis of finite state machines in unknown CMOS integrated circuits," *8th New Generation Scientist Conference*, Jena, Germany, pp. 45-50, Jan. 2007.

[11]M. Brutscheck, A. Kopp, E. Rosenfeld, St. Becker, M. Schwanecke, "Elektronische fokussierung von leistungsultraschall unter verwendung von field programmable gate array (FPGA)," *8th New Generation Scientist Conference*, Jena, Germany, pp. 204-205, Jan. 2007.

[12]A. Kopp, M. Brutscheck, St. Becker, E. Rosenfeld, "An ultrasound phased array transducer using FPGA technology," *International Congress on Ultrasonics, Viena*, Austria, p. 145, April 2007.

[13]M. Franke, A. Th. Schwarzbacher, M. Brutscheck, St. Becker, "Implementation of different square root algorithms," *China-Ireland International Conference on Information and Communications Technologies*, Dublin, Ireland, pp. 368-375, Aug. 2007.

[14]M. Franke, A. Th. Schwarzbacher, M. Brutscheck, "Implementation of different square root algorithms," *6th IEEE Electronic Circuits and Systems Conference*, Bratislava, Slovakia, pp. 103-106, Sept. 2007.

[15]M. Brutscheck, M. Franke, A. Th. Schwarzbacher, St. Becker, "Investigation of finite state machines in unknown CMOS integrated circuits," *14th Electronic Devices and Systems IMAPS CS International Conference*, Brno, Czech Republic, pp. 55-60, Sept. 2007.

[16]M. Franke, A. Th. Schwarzbacher, M. Brutscheck, "Implementation of different square root algorithms," *Research Report 2006/2007 University of Applied Sciences Merseburg*, pp. 93-97, November 2007.

[17]M. Pfau, A. Kopp, E. Rosenfeld, M. Brutscheck, St. Becker, "Ein ultraschall-phased-array für technische anwendungen," *Research Report 2006/2007 University of Applied Sciences Merseburg*, pp. 125-129, Nov. 2007.

[18]M. Brutscheck, A. Th. Schwarzbacher, St. Becker, "Systematic analysis of unknown integrated circuits - ein kooperatives promotionsverfahren," *Research Report 2006/2007 University of Applied Sciences Merseburg*, p. 90, Nov. 2007.

[19]M. Brutscheck, St. Berger, M. Franke, A. Th. Schwarzbacher, St. Becker, "Separation procedure for finite state machines in unknown CMOS integrated circuits," *9th New Generation Scientist Conference*, Köthen, Germany, Apr. 2008.

[20]M. Franke, V. Rusin, M. Brutscheck, R. Kasper, H. Mrech, U. Schmucker, "Multidomäne simulation einer antriebsmaschine," *9th New Generation Scientist Conference*, Köthen, Germany, Apr. 2008.

[21]M. Brutscheck, St. Berger, M. Franke, A. Th. Schwarzbacher, St. Becker, "Structural division procedure for efficient IC analysis," *Irish Signals and Systems Conference*, Galway, Ireland, pp. 18-23, June 2008.

[22]M. Franke, V. Rusin, M. Brutscheck, R. Kasper, H. Mrech, U. Schmucker, "Investigation of an electrical machine in multidomain environments," *15th*

Electronic Devices and Systems IMAPS CS International Conference, Brno, Czech Republic, pp. 408-413, Sept. 2008.

[23]M. Brutscheck, M. Franke, A. Th. Schwarzbacher, St. Becker, "Investigation and implementation of test vectors for efficient IC analysis," *15th Electronic Devices and Systems IMAPS CS International Conference*, Brno, Czech Republic, pp. 25-30, Sept. 2008.

[24]M. Brutscheck, M. Franke, A. Th. Schwarzbacher, St. Becker, "Investigation and implementation of test vectors for efficient IC analysis," *Electroscope Electronic Magazine*, Volume 2008.

[25]M. Franke, V. Rusin, M. Brutscheck, "Modellierung und simulation eines rolling rotor switched reluktanz motors," ASIM-Workshop Dresden, March 2009, (accepted for publication).